FPGA PASSPORT

Photo Booth Effects

Lorenzo Vigano & Diana Wofk

6.111 Fall 2016

Table of Contents

1 Project Abstract

The modern smartphone has made taking pictures extremely commonplace and easy to do. With the invention of social media networks like Facebook, Snapchat, and Instagram, it is very simple to share photos of yourself in amazing places doing incredible things. The majority of everyday life, however, is less amazing and more mundane. Our motivation for this project was to turn the mundane into the amazing.

The FPGA Passport will transport its user across the world by editing their face/body into exciting locations. Our project will assume that you are standing in front of a green screen. The FPGA will take in data from a camera pointing at you and will display the camera input on a monitor when a switch is flipped. You will then be able to press a button to begin image editing. The green screen background will be removed, and you will have the option to choose from a number of stored landmark backgrounds. Using chroma-keying, we will overlay your face on top of your chosen background.

You will also have the option of adding text and graphics to the image. Do you fancy yourself to be a king or queen? Try on the crown graphic and maybe even add some crazy facial hair! Have you always wanted to walk through a rain-forest? We have the perfect adventure-hat for you to try on. The FPGA Passport system will be able to place a crown on your head or gear you up for a tropical adventure. And if the placement is not quite right? Hat too low? Mustache too high? You will have the option of fine tuning the location of these overlayed graphics via button inputs. The composited image along with the optional text and graphics will be shown on the monitor.

When you are satisfied with the picture, you will be able to press a button to send the image data to a PC in uncompressed bitmap format. The selfie will be saved as a bitmap image on the PC, so that you, too, can brag about your travels to all of your friends.

2 Project Goals

Baseline Goals

(1) NTSC camera input to the Labkit

(2) User stands in front of green screen and flips a switch to display camera input
   (a) Processing begins: green screen detected and removed/replaced by user-selected background (user selects background via input buttons)
   (b) Composited image shown on monitor
   (c) User proceeds with image editing by using switches to select editing options and using buttons for customization (see expected goals)

(3) User will be able to toggle a switch to add text to the composited image
   (a) Default text will be "GREETINGS FROM ____ " -- these will be pre-programmed and will correspond to user-selected backgrounds; for example, if there is an image of the Eiffel Tower in the background, the blank in the default greeting will say "PARIS"
   (b) User will be able to move text around the screen using four button inputs

Expected Goals

(1) Background "destination" images stored in Flash memory; the FPGA is able to read these images out of Flash memory and display them on the monitor

(2) User will be able to type custom text and add it to the composited image

(3) User will be able to toggle a switch to add graphics to the composited image
   (a) Example graphics: crown, mustache, safari hat, and sunglasses
   (b) User will be able to move graphics using four button inputs

(4) User will be able to toggle a switch to add filter effects to the output image
   (a) Implemented filters will be selected via button inputs
   (b) Basic filters: sepia, negative / invert

(5) User will be able to enhance the composited image
    (a) User will be able to change the saturation and brightness of the image
    (b) Enhancements will be made visible on the monitor in real-time

Stretch Goals

(1) Image data transfer to PC in uncompressed bitmap format
(2) More advanced filters: sketch, blur, cartoon
(3) Face detection to assist in auto-placement of graphics
(4) Real-time processing on video input instead of on static image

3 System Block Diagram

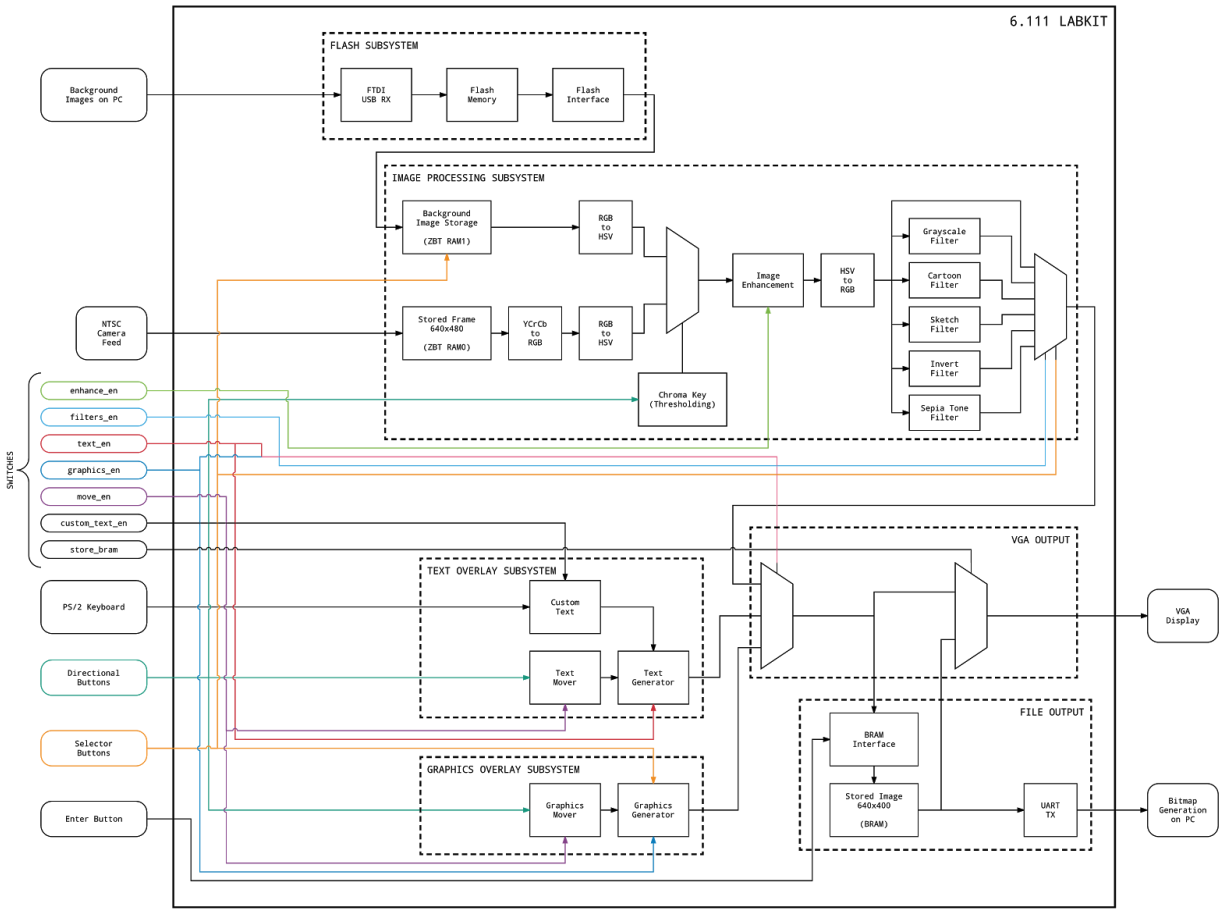


Figure 1: High-Level Block Diagram

The figure above shows a high-level block diagram of our system. Flash memory and the flash interface both run on the 27 MHz system clock. Custom text input is also processed at 27 MHz. All image processing, text generation, graphics generation, and pixel output are done in sync with VGA timing; in this project, for 800x600 @ 60 MHz, a 40 MHz clock is used. Data transmission from the FPGA via UART is done at a baudrate of 115200.

4 Subsystems

4.0 Main FSM - Diana

Module(s): `main_fsm`, `zbt_6111_sample`, `debounce*`, `display_16hex*`

The **`main_fsm`** module defines the state transitions for the overall system. The finite state machine consists of the following 6 states:

(1) `FSM_IDLE` - the default idle state; camera input not being processed

(2) `SEL_BKGD` - camera frame displayed on screen; user able to select background

(3) `COLOR_EDITS` - user able to enhance image and/or apply filter effects

(4) `ADD_EDITS` - user able to overlay text and/or graphics over image

(5) `SAVE_TO_BRAM` - edited image saved to BRAM memory and then displayed

(6) `SEND_TO_PC` - stored image transferred to PC in uncompressed bmp format

State transitions occur on edges of debounced user inputs. Both button and switch inputs are debounced using the **`debounce`** module, and their rising/falling edges are used to advance through the state machine in a linear fashion:

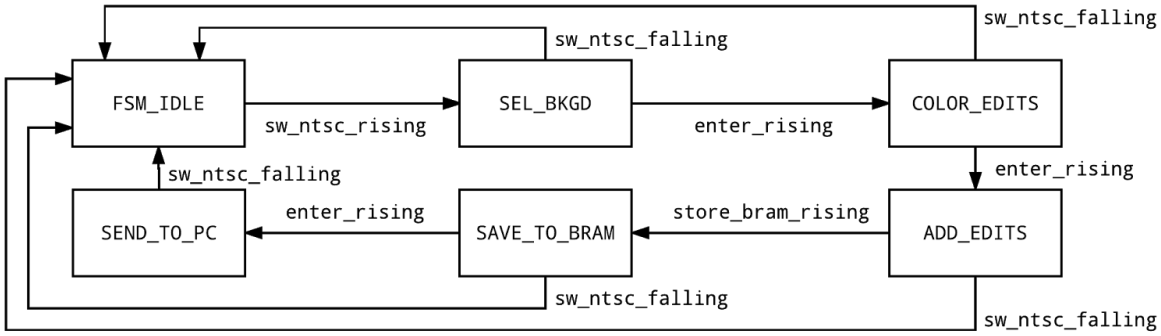


Figure 2: Main Finite State Machine

The **`main_fsm`** module is instantiated in the **`zbt_6111_sample`** top module that also instantiates all other major subsystems. The **`main_fsm`** module has only one output, `fsm_state[2:0]`; this signal is displayed as the leftmost nibble on the hex display.

4.1 Camera Input - Lorenzo

Module(s): `video_decoder*`, `ramclock*`, `ntsc2zbt*`, `vram_display*`, `zbt_6111*`, `ycrcb2rgb*`

The majority of the code for NTSC camera interfacing was provided by the 6.111 lab staff, though it required modifications to add color. The **`video_decoder`** and **`ntsc2zbt`** code take in data from the NTSC and convert it into easily usable data. By synchronizing with the system clock and converting the data from YCrCb to RGB, these modules make taking data from the camera significantly easier.

After this, the data is then stored in ZBT. The original code was set up to store two grayscale pixels in one space in memory, however, color was needed for this project. Therefore, the code was modified to store one 24-bit pixel in one memory location. Because we didn't need live feed, we stored a static image in ZBT.

Before displaying this image through VGA, the color had to be converted to RGB. This was also done with staff provided code; the process and intricacies of color spaces are described in the following section.

After this conversion, the **`vram_display`** module then reads from ZBT and fetches the appropriate pixel in memory for a given `vcount` and `hcount`. This pixel is then given to the VGA and then displayed on the screen.

4.2 Color Space Conversion - Diana

Module(s): `rgb2hsv*`, `hsv2rgb`

There are several ways to represent color; commonly used color spaces include:

- **RGB** - in which red (R), green (G), and blue (B) light are added to produce colors
- **YCrCb** - in which colors are represented as brightness and two color difference components; Y is the luma (brightness) component, Cb is the blue-difference (B-Y) component, and Cr is the red-difference (R-Y) component
- **HSV** - in which colors are represented by hue (H), saturation (S), and value (V)

Color space conversion plays a significant role in our project as it simplifies chroma-key compositing. The HSV color space is ideal for chroma-keying since it allows for easier thresholding; the hue threshold can be set to the chroma key, which in our case is the color green, and the threshold for keying can then be adjusted over saturation and value. This results in more accurate compositing, where slight deviations in the brightness of the green screen backdrop (e.g. due to lighting conditions, shadows, etc) do not compromise the quality of the composited image.

Most graphics, including the background images used in our project, are stored as RGB values. Furthermore, the decoded input video frame is converted from YCrCb to RGB. In order for chroma-key compositing to be done in the HSV color space, conversion from RGB to HSV is needed; this is implemented in the staff-provided **`rgb2hsv`** module. The module uses two IP core dividers and is pipelined with 23 stages.

Since the output after compositing is in the HSV color space, it needs to be converted back into RGB before being sent to the VGA display. The **`hsv2rgb`** module carries out this conversion using only integer math[1] in a pipelined manner with 10 stages.

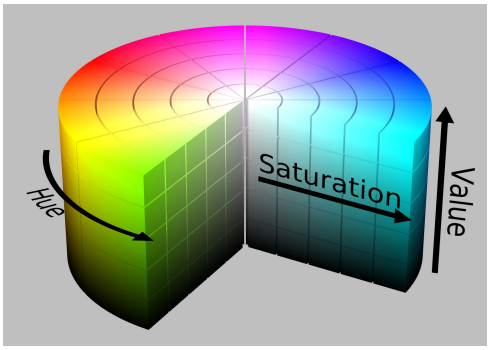


Figure 3 illustrates how hue, saturation, and value are defined in the HSV color space. Hue ranges from 0 to 360, while saturation and value each range from 0 to 100. In the color space conversion modules, these are all normalized to be in the 0 to 255 range.

Figure 3: HSV Color Space

[1] C Implementation: http://web.mit.edu/storborg/Public/hsvtorgb.c

4.3 Image Transfer to Labkit - Lorenzo

Module(s): `usb_input*`

Program(s): `serial1.py, JPEGtoCOE.m, converter.py`

Through a combination of MATLAB and Python scripts, images were converted from JPEG to a format that was compatible to be sent from a PC to the Labkit via FTDI UM245R USB.

MATLAB provided a great built in tool to analyze images. After simply importing the image and parsing it with the built-in function `imread()`, which creates matrices of red, green, and blue values for each pixel, the script iterates through all pixels and writes these values as binary values into a text file. The script was designed to convert the picture to 16-bit color (5 bits of red, 6 bits of green, and 5 bits of blue). 16-bit color was chosen due to space constraints within the flash memory.

The MATLAB script generates a text file that is not easily parsed by **`serial1`**, which therefore created the need for a small Python program (**`converter`**) to convert this data into data that could be easily sent to the FTDI.

After running this process four times (for all background images), all the data was compiled into one large file that was then sent by **`serial1`** to the FTDI UM245R.

4.4 Image Storage on Labkit - Lorenzo

Module(s): `usb_input*, flash_write, vram_display, zbt_6111_sample`

As mentioned above, data was sent to the labkit from the PC via the FTDI UM245R. After data was received, it was stored in a small buffer, and then the staff provided **`usb_input`** would output the data to the main **`flash_write`** module.

Every time a new piece of data was available, a ready signal was given by **`usb_input`**. Data from this module, however, only came one byte at a time, which was only one half of the pixel data. At each ready signal, one piece of data was stored into the upper half of a 16-bit register and then it was completely filled at the next ready signal. Once full, this data was the passed into a large FIFO, which interfaces with the Flash memory. After

storing the pixel in the FIFO, the register was cleared and the next data set was ready to be received from the **`usb_input`**.

The aforementioned FIFO was used to cross the clock domains and help deal with the unpredictable nature of Flash, which will be discussed in the next section.

Flash Memory

As mentioned above, Flash memory, while incredibly useful, is finicky and definitely challenging to interface with at times. Due to memory constraints, it was impossible to also store the background images in BRAMS, so they were stored in the large Flash memory bank. In total, five images were stored in flash (four backgrounds 640x480 and one start-up 640x360). Each memory location (1459200 total used) holds 16 bits of information which is why the images were originally converted to 16-bit color.

Flash, unlike some other systems, does not have a ready signal, but rather a busy signal. Whenever flash is asked to do a specific task, it asserts its busy signal and then carries out the task. Once completed, it lowers the busy signal, conveying that it is ready for the next instruction. The difficulties here come in that the Flash in not often consistent. Sometimes it takes the same time to write a pixel to memory and sometimes not. This was the primary need for the FIFO. Because it was not certain when the flash would be ready for the next piece of data, the FIFO acted as a great buffer and would given the Flash data only when the Flash was ready.

There was also a slightly complicated reset mechanism, which took a while to debug, this will be covered further in the Challenges section. Resetting was crucial to correct writing, but it was very important that once the data was written correctly, that it would not be overwritten or reset. The Flash reset did not like being hardcoded to a zero, so it was set to a very complicated series of switches and button combinations that would never be able to happen on accident.

ZBT Memory

While Flash is very useful in that it has the capability to store large amounts of data even when power has been turned off, it is unfortunately very slow. This is a problem when displaying the images. The background images were therefore cached to ZBT. There are two ZBT RAMs on the labkit. The first RAM (ram0) was used the store camera input data, as mentioned in Section 4.1, and the second RAM (ram1) was used to cache a selected background image. The background image was selected via button inputs where 0, 1, 2, and 3 corresponded to Paris, Rome, Amazon, and London, respectively.

Due to a combination of different clock domains between ZBT and Flash, and the often unpredictable nature of the Flash busy signal, a FIFO was used to attempt to deal with these issues. This, however, introduced other complications. It was still difficult to properly assert the write and read enable signals. Therefore, another system was devised.

Because the Flash was operating at 27 MHz and it takes several of these clock cycles to retrieve data from flash, it was certain that by testing the falling edge of busy on each clock cycle, using the ZBT clock, we could accurately capture the correct data (Flash reading code is in top file of **`zbt_6111_sample`**). When the falling edge of busy is detected, data on the Flash data-out wire is stored into a register that is then passed to the ZBT input, the ZBT write enable is asserted, and the ZBT write address is increased. During the caching phase, the write enable signal is constantly high. It is the address and data that change at the falling edge of busy. After the image has been loaded from flash to ZBT, the write enable signal is deasserted and the `vram_display` module gains control of addressing.

At this point the selected image has been loaded into ZBT, but now it needs to be read and fed to the VGA. Using **`vram_display`**, the address for the desired pixel is determined using a given `hcount` and `vcount`. Like one reads, left to right then top to bottom, the pixels were stored such that the top left pixel was in the first memory address, and thus the last pixel from the bottom right corner was stored in the last memory location.

Using this storage scheme, **vram_display** accessed vcount*640+hcount to obtain the specified pixel. Because of a two cycle delay in accessing things from ZBT memory, **vram_display** forecasted ahead in order to offset this delay. After receiving the pixel data, it then feeds this information into the VGA outputs. Taking the bits and placing them in the higher order locations to move the 16-bit color into the 24-bit output.

4.5 Chroma-Key Compositing - Diana

Module(s): chroma_key, pixel_sel

Chroma-key compositing consists of two steps: (1) chroma-key detection, and (2) compositing. The **chroma_key** module carries out chroma-key detection. It takes in 24-bit HSV as input and determines whether the HSV values fall into pre-defined threshold ranges. Local parameters within the module define separate nominal thresholds for hue, saturation, and value, as well as a range parameter for specifying the tolerance with which input HSV values are considered to match thresholds. Default settings for these nominal thresholds were chosen based on preliminary testing during system integration; the hue threshold, for example, was set to bright green. However, to better account for fluctuating lighting conditions, and to offer more customizability regarding which chroma key is used (green screen, blue screen, etc), all thresholds are user-adjustable. The **chroma_key** module contains the logic controlling H, S, and V threshold adjustment via the directional user input buttons: pressing the up/down buttons while leaving the left/right buttons untouched adjusts the hue threshold; pressing the up/down buttons while holding down the left button adjusts the saturation threshold; pressing the up/down buttons while holding down the right button adjusts the value threshold. The range used in thresholding over hue and saturation can also be adjusted by pressing the up/down buttons while holding down *both* the left and right buttons. The effects of adjusting thresholds are visible on-screen in real-time because the changes are immediately reflected in how the system detects the chroma key color.

Chroma key thresholds are adjustable only when the main FSM is in the SEL_BKGD state, i.e. the state in which the user selects a background image choice. The compositing of the camera input frame with the selected background image is done in the **pixel_sel**

module, where the chroma_key_match signal is used a selector; if chroma_key_match is asserted, the composited pixel will be the background image pixel; otherwise, it will be the HSV pixel from the camera input frame. The 24-bit HSV output of the chroma-key compositing subsystem is then passed an input to the image editing subsystem.

4.6 Image Editing - Diana

The image editing subsystem contains several modules that implement various types of image processing. The modules are grouped in two categories: enhancement and filters.

Image Enhancement

Module(s): enhance

The **enhance** module allows the user to adjust the saturation and brightness of image pixels and view the changes on the screen in real time. Image enhancement is enabled or disabled using the enhance_en switch. When enhancement is enabled and the main FSM is in the COLOR_EDITS state, the four directional buttons inputs control saturation and brightness offsets. Pressing the up/down button increments/decrements the saturation offset, while pressing the right/left button increments/decrements the brightness offset. On every clock cycle, the offsets are applied to the original S and V values of each input HSV pixel, and an output pixel is generated with the modified S and V values. Each output pixel is then converted to RGB by the **hsv2rgb** module before being displayed via VGA, allowing the user to observe the saturation and brightness adjustment in real-time. Saturation and brightness can be adjusted simultaneously, and can also be reset to original values by pressing all four directional buttons together.

The set of figures on the next page demonstrates image enhancement:

Figure 4: Original Image Pre-Enhancement

Figure 5: Increased Saturation

Figure 6: Decreased Saturation

Figure 7: Increased Brightness

Figure 8: Decreased Brightness

Filter Effects

> **Module(s)**: `filters`, `sepia`, `invert`, `grayscale`, `cartoon`, `line_buf`, `sobel_op`, `edge_det`, `gaussian`

The **`filters`** module instantiates four filter modules and also contains logic to determine which filter the user has selected. Filter effects are enabled or disabled using the filters_en switch. When filter effects are enabled and the main FSM is in the `COLOR_EDITS` state, the four selector buttons (0, 1, 2, and 3) are used to select a particular filter. The mapping between selector buttons and filter effects is as follows:

> 0 : `SEPIA` - a simple filter that applies a sepia tone to create a vintage effect
> 1 : `INVERT` - a simple negative filter that inverts colors
> 2 : `SKETCH` - a filter effect that performs edge detection on the original image
> 3 : `CARTOON` - a combination of edge detection, bur, and conversion to 8-bit color

The default effect that is applied when filter effects are first enabled is `GRAYSCALE`. Afterwards, the system remembers which filter the user has selected. The `GRAYSCALE` effect can then be re-applied by pressing all four selector buttons together.

The **`filters`** module takes 24-bit color RGB as input and passes the input through all 4 filter submodules. It then selects the 24-bit RGB output pixel based on the selected filter.

Implementations of various filter effects are described below.

Grayscale

The **`grayscale`** module converts input RGB values to a single 8-bit monochrome value. The 24-bit output pixel from this module has its R, G, and B values equal to this mono-chrome value. Conversion to grey color is based on the NTSC formula for luminance:[2]

$$Y = 0.2989R' + 0.5870G' + 0.1140B'$$

[2] Color FAQ: http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html

where R', G', and B' are gamma-corrected components. Our Verilog implementation assumes that R' = R, G' = G, and B' = B, i.e. gamma correction is not taken into account.

The **grayscale** module is pipelined with 3 stages.

Sepia

The **sepia** module converts 24-bit RGB color input to sepia and output pixels with 24-bit sepia-tone RGB values. The algorithm used to compute sepia-tone values has two steps:[3]

(1) Compute intermediate RGB values by taking integer values of these sums:

$$tR = 0.393R_{in} + 0.769G_{in} + 0.189B_{in}$$
$$tG = 0.349R_{in} + 0.686G_{in} + 0.168B_{in}$$
$$tB = 0.272R_{in} + 0.534G_{in} + 0.131B_{in}$$

(2) Set RGB values of the output pixel according to these conditions:

$$\text{if } tR > 255, \text{ then } R_{out} = 255; \text{ else } R_{out} = tR$$
$$\text{if } tG > 255, \text{ then } G_{out} = 255; \text{ else } G_{out} = tG$$
$$\text{if } tB > 255, \text{ then } B_{out} = 255; \text{ else } B_{out} = tB$$

The **sepia** module is pipelined with 4 stages.

Invert / Negative

The **invert** module implements a simple negative filter, in which light areas of the image appear dark and dark areas appear light. A negative filter also reverses colors into their complementary colors; red appears as cyan, green appears as magenta, and blue appears as yellow. The **invert** module takes in 24-bit RGB color as input every clock cycle and outputs an inverted 24-bit RGB pixel on the next clock cycle, where:

$$R_{out} = 255 - R_{in}$$
$$G_{out} = 255 - G_{in}$$
$$B_{out} = 255 - B_{in}$$

[3] http://www.dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image

2D Convolution (3x3)

Implementing more advanced filters requires performing 2D convolution on the input image. The two remaining filters in our system carry out 3x3 2D convolution.

2D convolution is the process of adding each element of a matrix to its local neighboring elements, weighted by a given kernel. 3x3 convolution of a matrix with a 3x3 kernel involves (1) flipping the kernel both vertically and horizontally, (2) overlaying the kernel over the matrix, (3) multiplying the matrix elements by overlapping kernel values, and (4) summing the multiplied values to obtain the output value for the center element in the matrix. An example is shown in Figure 9.

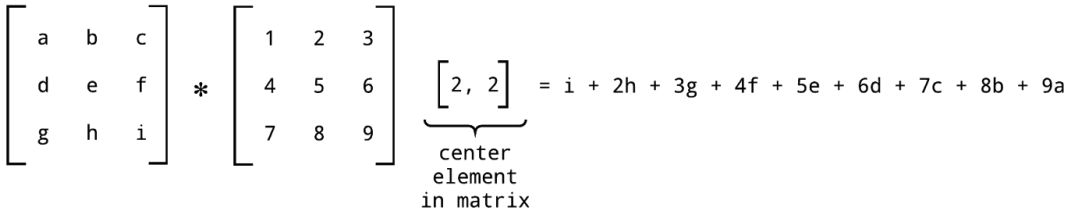


Figure 9: 2D Convolution

Performing convolution requires two copies of data - one that is used in the convolution calculations and one that is overwritten. Since the amount of internal memory available on the labkit FPGA is so limited, it is unfeasible to store a duplicate copy of the image that is being filtered. Instead, line buffers are used to store the rows of image data that are immediately used in the convolution calculation. For 3x3 convolution, two consequent rows of data and three data elements of the third consequent row are needed. In our design, the **`line_buf`** module creates 2 line buffers that are each as long as the image width as well as an additional 3-element buffer. These buffers are actually shift registers through which image data passes; they act as latency pipes for incoming streams of pixels and simulate sliding a 3x3 window in raster scan across the input image. Line buffers and the 3x3 sliding window are illustrated in Figure 10.

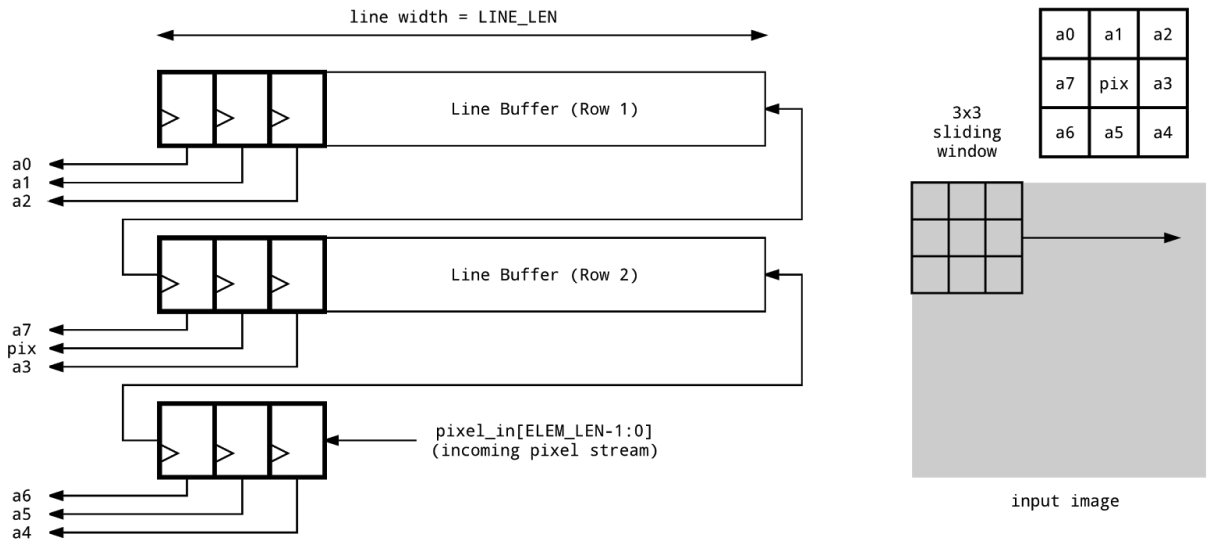


Figure 10: Line Buffers and Sliding Window

The **`line_buf`** module outputs nine values that correspond to the matrix of values overlaid by the 3x3 window; this is the matrix to which Gaussian kernels and Sobel operators will later be applied. The **`line_buf`** module is parameterized by both line width `LINE_LEN` (i.e. the desired length of the line buffer) as well as by element width `ELEM_LEN` (i.e. the size of each element to be stored in the line buffer).

Gaussian Blur

The **`gaussian`** module applies a 3x3 Gaussian kernel to an input 3x3 matrix of pixel values. Since each pixel value actually consists of three distinct values (namely, RGB), processing is done on each channel separately. Furthermore, since blurring is being done with the end goal of creating a cartoon effect, 8-bit color is used. The **`gaussian`** module therefore takes nine 2-bit R values, nine 2-bit G values, and nine 1-bit B values as inputs. To each of these 3x3 matrices, the 3x3 Gaussian kernel shown to the right is applied. The **`gaussian`** module is pipelined with 3 stages and outputs an 8-bit color blurred RGB value.

$$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

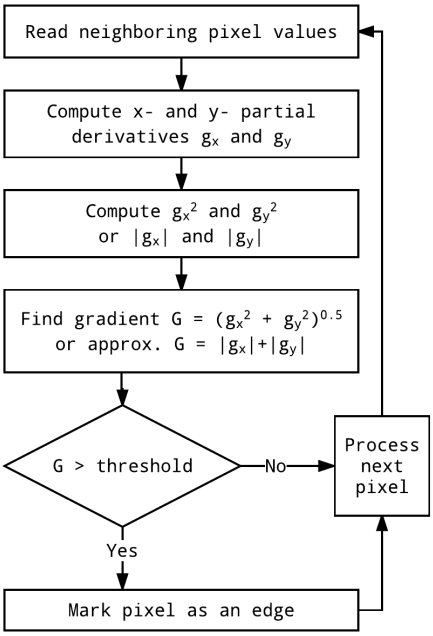
Sobel Edge Detection

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$A = \begin{bmatrix} a0 & a1 & a2 \\ a7 & pix & a3 \\ a6 & a5 & a4 \end{bmatrix}$$

$$g_x = G_x * A$$

$$g_y = G_y * A$$



The **sobel_op** and **edge_det** modules implement simple edge detection on the input image. Like the **gaussian** module, the **sobel_op** module takes in a 3x3 matrix of pixel values as input; for edge detection, grayscale input is used, so only one matrix consisting of nine 8-bit monochrome values is processed. The Sobel operator calculates the gradient in a region of pixels by applying two convolutional masks to the input matrix and calculating partial derivatives, g_x and g_y, which detect gradients in the x- and y- directions respectively. After calculating the gradient, the **sobel_op** module passes the 16-bit value to the **edge_det** module that then compares the gradient against a predefined threshold and determines whether the pixel of interest corresponds to an edge. The computation of partial derivatives is shown in Figure and the overall flow for Sobel edge detection is summarized in Figure. The **sobel_op** module is pipelined with 4 stages, and the **edge_det** module introduces an additional 1-cycle delay.

Figure 11 (top): Sobel Operator, **Figure 12** (bottom): Sobel Edge Detection Flow

Sketch & Cartoon

The **cartoon** module instantiates both the Sobel edge detection modules and the Gaussian blur module, as well as all of the needed line buffers. The module takes both 24-bit RGB color and grayscale as input. It implements a sketch effect by applying just edge detection to the input and a cartoon effect by combining the outputs of both edge detection and 8-bit color Gaussian blur. The outputs of the cartoon module include

`rgb_edge[23:0]`, a sketch effect output pixel, and `rgb_cartoon[23:0]`, a cartoon effect pixel. Since edge detection is involved in both filter effects, the **`edge_det`** module distinguishes between the two effects by using different thresholds for marking edge pixels. The threshold used for the sketch effect is lower than that used for the cartoon effect, in order for the sketch output to appear more "detailed."

The set of figures below demonstrates the filter effects described in this section:

Figure 13: Original Image Pre-Filtering

Figure 14: Grayscale

Figure 15: Sepia

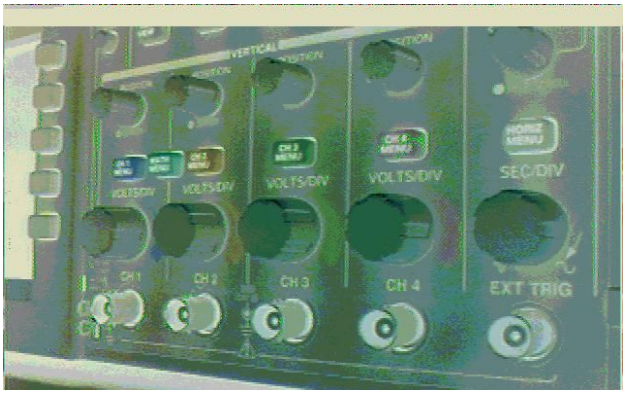


Figure 16: Negative / Invert

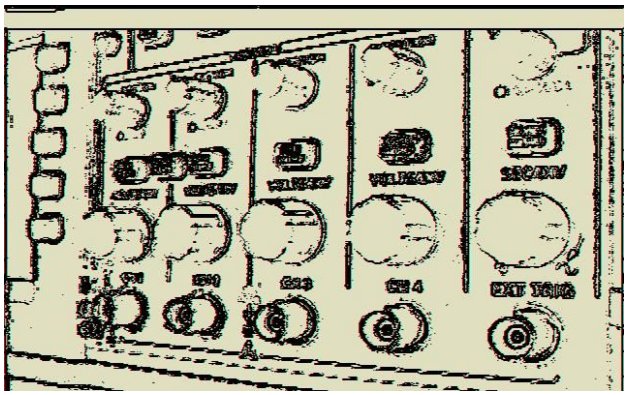


Figure 17: Sketch (Edge)

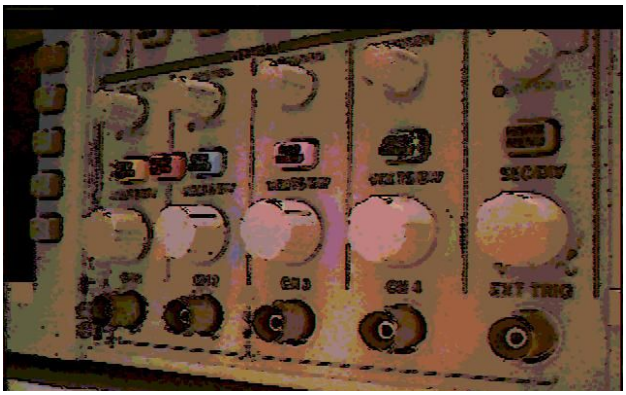


Figure 18: Cartoon (Edge + Blur)

4.7 Text Generator - Lorenzo

Module(s): `stringmaker`, `text_gen`

These modules were used to create sprites of letters to be displayed on the screen. This module, however, does not directly display the pixel on screen, but rather when given an `hcount`, `vcount` and coordinates, it determines whether or not there should be color displayed.

There are two internal components to this system. There is block ROM interface module, **`text_gen`**, which handles one letter at a time and pulls data from the font ROM initialed with the `letters.coe` file (data for all letters of alphabet and a space; the letters are of fixed size and font.). And also a module, **`stringmaker`**, which instantiates **`text_gen`** and handles inputs such as custom text versus default text. Given the background selection, the default text generator section of the module outputs the message "GREETINGS FROM [background location]", however, if the custom signal is high, then data on the custom text input is taken.

The process of stringing a word together was more complicated than anticipated but after a few iterations, a highly functioning system was developed. The **`text_gen`** module takes in both a `vcount` and `hcount`, but also and offset which allow for moving text. Initially, these default to the coordinates (0,0), but the user has the ability to move the text around the screen by use of buttons on the labkit. This functionally was exploited to write multiple letters at the same time. When the top line of the first letter is being written,

stringmaker instructs **text_gen** to access and then display the top line of that letter. Then, after hcount moves the width of a letter (15 pixels), **stringmaker** tells **text_gen** to now fetch the top line of the next letter, not at the original location, however, but instead at a location 18 pixels away from the previous letter. These 18 pixels provide space for the previous letter and a small buffer in-between. Now on the same at the same vcount the top part of the letters are all displayed. When vcount increases, this process is repeated, but instead, **text_gen** fetches the next line of each letter.

When the user turns the custom text switch high, the default text disappears and the module waits for input from the user. After raising the switch, the user can then type what they want on the keyboard and when they are done they hit the enter button on the Labkit. Upon hitting this button, the Custom Text Input modules are initiated. They will be described in the next section. They output to the **stringmaker** a ready signal, an array of bytes that each corresponds to an ascii character, and the number of characters that were entered by the user. Much like how **stringmaker** handled default text, it then calls **text_gen** which accesses the selected letter in memory and outputs whether a given location is within the sprite boundary.

4.8 Custom Text Input - Diana

Module(s): custom_text, ps2*, ps2_ascii_input*

The **custom_text** module acts as a wrapper for the staff-provided **ps2_ascii_input** and **ps2** modules that together process PS/2 keyboard input to the Labkit. The **ps2_ascii_input** module outputs 8-bit ASCII values along with accompanying ready pulse signals. These ASCII bytes correspond to the characters that the user has typed on the keyboard. When our system's main FSM in the ADD_EDITS state and custom text has been enabled via the custom_text_en switch, the **custom_text** module continuously checks whether a char_rdy signal has been asserted by the **ps2_ascii_input** module. This indicates that another incoming character byte has been processed, and the **custom_text** module then stores this new ASCII value in a character array. The array is limited in size, with max length being a parameter set to a default value of 20. This means that the first 20 characters that the user types on the keyboard will be read by the

custom_text module; all following keyboard input will be ignored. The length parameter can be easily modified if user wishes to process longer or shorter text.

ASCII values are stored in the character array from left to right, i.e. the first character entered is stored as the most significant byte in the array. The character array is finalized when the user presses an enter button on the Labkit after having typed the desired text. At this moment, the **custom_text** module also determines the total number of characters that the user has entered and asserts a `char_array_rdy` signal indicating that the character array is ready to be processed by the text generation subsystem.

4.9 Graphics Generator - Lorenzo

Module(s): `graphicsmaker`

This module is similar to the Text Generator module, however, it has fewer states. As opposed to 27, it has four. One for a crown to match the London setting, a mustache for Paris, sunglasses for Rome, and an adventure hat for the Amazon.

Like the Text modules, the Graphics generator has a COE file loaded into a block ROM. **graphicsmaker** accesses the chosen graphic, and like the text, determines whether a specific location on the screen should have color or not.

Because this COE file was given radix 2, other colors had to be hardcoded into the output. **graphicsmaker** could tell if there should be color and then after that, if it was within a specific section of them memory, it was assigned different colors. For example, the top of the crown was made red by assigned the color variable to be red above a certain line in memory and gold below that line.

Unlike the Text modules, **graphicsmaker** can only make on image at a time. Though a system very similar to that used for text could be implemented to create multiple graphics at once.

On the next page, there are four photos that show a few of the many different combinations that a user could create. For example, they could have custom text with a mustache graphic in Paris.

Figure 19: Paris

Figure 20: Rome

Figure 21: The Amazon

Figure 22: London

4.10 VGA Output - Diana

Module(s): `pixel_sel`, `mover`

All of the image processing done in this project is done in real-time; intermediate images (e.g. between image enhancement and applying filter effects) are not stored anywhere in memory. Only the initially-taken NTSC camera input frame is stored in ZBT memory. Processing is continuously done on a pixel-by-pixel basis, and what is displayed on screen is actually the same camera input frame being processed over and over with the same settings (threshold/enhancement settings, background/filter selections, etc) and the same text/graphics overlay. Pixel display is largely controlled by the VGA output subsystem, which consists primarily of one major pixel selector module (`pixel_sel`) that

instantiates all of the image processing modules (color conversion, enhancement, filters) as well as the text and graphics modules. The `pixel_sel` module also interconnects all of the instantiated processing modules, resulting in a somewhat linear processing path:

> stored YCrCb pixel from NTSC input → YCrCb2RGB → RGB2HSV → chroma-key compositing → image enhancement → HSV2RGB → filter effects

The pixel output from the filter effects subsystem, and therefore of the image processing subsystem, is `pixel_filtered[23:0]`, one of the 3 pixel outputs that can get displayed.

In addition to instantiating the text and generator modules, the pixel_sel module also instantiates two versions of the mover module, one of which is used for controlling the movement of displayed text and, the other, of displayed graphics. The mover module generates x- and y- coordinate offsets given user button inputs. When the main FSM is in the ADD_EDITS state, the four directional buttons can be used to adjust the position of the top left corner of text/graphics sprite on the screen. The directional buttons control text movement when the move_en switch is low and they control graphics movement when the switch is high. Position offset values are updated on the falling edge of the VGA timing signal vsync. The **`pixel_sel`** module connects the text and graphics mover modules to the text and graphics generator modules, resulting in two additional distinct subsystems (text overlay and graphics overlay). In each of these subsystems, sprites are generated on a white background, and the respective pixel outputs are `text_gen_pixel[23:0]` and `graphics_gen_pixel[23:0]`.

In determining which of the three aforementioned pixel outputs actually gets displayed on screen, the **`pixel_sel`** module checks whether the `text_en` and `graphics_en` switches have been turned on (thereby enabling text/graphics overlay). If both are enabled, pixels are selected for display in the following order of priority:

> (1) `text_gen_pixel` (2) `graphics_gen_pixel` (3) `pixel_filtered`

The selected pixel output is assigned to `vga_rgb_out[23:0]`. This signal is registered and connected to top-level VGA pixel output as long as the main FSM is not in the

FSM_IDLE, SAVE_TO_BRAM, and SEND_TO_PC states. In those states, either a blank screen or data stored in BRAM is displayed.

The **pixel_sel** module also delays VGA timing signals to ensure that they are in sync with the VGA pixel output. Since all image processing is done on the fly, the delays from all of the pipelined processing modules accumulate to a considerable total delay. Furthermore, the delay depends on type of filter that the user has selected, since more advanced filters require more processing time. The **pixel_sel** module uses three shift register that are each large enough to delay the VGA hsync, vsync, and blank signals by the longest possible delay needed; the module then checks which filter has been selected and outputs the appropriately delayed timing signals.

4.11 Image File Output to PC - Diana

Module(s): bram_ifc, uart_tx, zbt_6111_sample

Program(s): uart_rx.py, bin2bmp.c, save_bmp.sh

Storage in BRAM

When the main FSM is in the ADD_EDITS state and the user has finished editing the image, the user can trigger a transition to the SAVE_TO_BRAM state, in which a cropped copy of the edited image is stored in block memory. Due to the limited amount of internal memory available on the FPGA, the image is cropped from 640x480 to 640x400 and converted from 24-bit color to 8-bit color before being saved in 256000x8-bit single-port BRAM. The **bram_ifc** module contains an FSM that controls writing to and reading from the instantiated BRAM. A user input switch enables writing to BRAM, which initiates the linear progression through the FSM states: from the BRAM_IDLE state, the BRAM FSM transitions to the CAPTURE_FRAME state, in which it waits until the VGA hcount and vcount signals correspond to the top left corner of the output image before beginning to store pixel values. Image data is written to BRAM in the WRITING_FRAME state, and the FSM transitions immediately to the READING_FRAME state after the entire image has been saved in BRAM. The stored data is then continuously read out and displayed via VGA. Turning the user input switch off resets the BRAM FSM and allows stored data to be overwritten with new data the next time the switch is raised.

Transfer to PC

When the main FSM is in the `SAVE_TO_BRAM` state, the user can initiate image data transfer to the PC by pressing the enter button. The BRAM state will remain `READING_FRAME`, except now the read address will be the `tx_counter` corresponding to the number of RGB values that have been transmitted from the FPGA to the PC. `tx_counter` is incremented on every third clock cycle, to allow enough time for the 24-bit RGB values to be transmitted as three 8-bit values. Image data is transferred via UART.

The `uart_tx` module implements a simple parameterized UART transmitter with a default baudrate of 115200. By default, the transmitter sends 1 start bit, 8 data bits, and 2 stop bits per data byte. The TX clock is generated from the 40 MHz `clk` signal used throughout our overall design. The `uart_tx` module accepts 8-bit data and a TX enable signal as inputs. When the TX enable signal is asserted, the input data is stored in a shift register, along with the preceding start bit and succeeding stop bits; the length of the shift register used corresponds directly to the total number of bits that will be transmitted per data byte (in this case, 1+2+8 = 11). Then, on each rising edge of tx_clk, one bit is transferred - beginning with the start bit, proceeding with the data bits from LSB to MSB, and concluding with the stop bit(s). If the TX enable signal remains asserted and a new byte of data is waiting to be sent after the current byte has been transmitted, the new input data is stored and subsequently transmitted. The transmitted bit signal is connected to a user output wire.

On the receiving PC end, a Python program running on the PC (`uart_rx.py`) receives the transmitted data and temporarily stores it in a FIFO buffer. After all of the image bytes (3*640x400 = 768000) have been received, the buffered data is written in binary mode to a .bin file. A C program (`bin2bmp.c`) that structures binary data into a bitmap file then runs and outputs a .bmp file. The end result is an uncompressed bitmap image stored on the PC. For convenience, a shell script (`save_bmp.sh`) that calls the Python program and the C program in the correct sequential order has also been created.

Since no issues arose in transferring data during testing, the Verilog implementation of the UART transmitter used in this project does not support RTS/CTS flow control.

5 Testing & Debugging

In addition to common testing and debugging that is implicit in any project, we spent the majority of our time debugging issues with timing. Due to the multiple clock domains involved, it was very important that we had a clear approach to integrating and testing our modules. We chose to build our project in a very modular way and took great care to confirm the functionality of one module or subsystem before moving to another. The hex display and LED output were used very often to ensure that data was being manipulated in the way we intended for it to be. For example, whether data was properly being stored in a specific memory location or when data was being altered for filter effects.

When anomalies occured, and we suspected bugs, we began by passing out values to the hex display and LEDs to be sure that our logic was behaving how we intended it to be behaving. Generally, we took a top down approach: making sure earlier logic functioned correctly before attempting to debug the final output. If we could ensure all early code and logic worked, then we could deduce where our error was occurring.

Testbenches were used for a few of the color space conversion modules and filter modules. Both the `rgb2hsv` and `hsv2rgb` modules were verified in simulation with simple testbenches that tested several input RGB / HSV values. A testbench was also used to verify the `sepia` filter module, primarily because the conversion from RGB to sepia tone is formulaic in nature and easily testable in simulation. Testbenches were not used for more advanced filters as careful modular design resulted in expected functionality with minimal debugging, which eliminated the need for simulation.

Testbenches were also not used for testing interfaces to Flash memory or the NTSC due to their behavior being difficult to simulate. The Challenges section will discuss the intricacies of a particularly frustrating bug in the flash memory code.

6 Challenges

One of the main reasons that we chose to use Flash memory was due to its ability to hold large amount of data for a long time, regardless of whether the labkit remained powered. There was, however, a strange bug in the staff provided code where in which flash would reset upon the FPGA being reprogrammed. And though it took a while to find, it was eventually corrected, however, not before losing about five days.

Even after this was fixed, Flash memory still appeared to occasionally exhibit illogical behavior, such as varied read time lengths, and special care was taken when interfacing the Flash with any other system, such as the FTDI USB and ZBT memory. Flash was significantly easier to read from than to write to, therefore once Flash was correctly written to, the remainder of the background image process, such as reading from Flash and caching to ZBT, seemed easier in comparison.

Since Flash gave us so much trouble, when it became evident that caching the image from Flash to ZBT wasn't working properly, we initially assumed that the problem was with Flash. However, this turned out to be an incorrect assumption. We then began to suspect problems in reading out of ZBT.

Our project uses VGA timing, but when originally coding this part of the project, all timing was based of the XVGA clock. ZBT memory accessing when reading and displaying was based off of `hcount` and `vcount` (described in detail in Section 4.4 - ZBT Memory). XVGA timing and VGA timing are different because of the different resolutions they support (800x600 for VGA and 1024x768 for XVGA) and when we integrated this module with the project as a whole, the image did not display correctly. However, once we realized this, we adjusted the code for VGA timing and devised a more straightforward addressing system, and then the image displayed perfectly.

We also encountered difficulties with internal memory throughout our project. The Labkit contains limited BRAM resources: 144 18-Kbit blocks, for a total 2952 Kbits. When storing images in 8-bit color, it seems as if this amount of internal memory is enough to

store a 640x480 image (8*640x480 = 2457.6 Kb). However, we found out that generating a BRAM with a width of 8 and a depth of 307200 resulted in overmapped BRAM resources, indicating that the depth of the generated BRAM was likely increased to the nearest power of 2, i.e. 2^19 = 524288. To resolve this issue, we decided to crop our output image and only store 640x400 pixels, which required up to a maximum depth of 2^18 = 262144. Reducing the amount of BRAM used for storing our output image also left us with enough resources for our font ROM, graphics ROM, and line buffers used in 2D convolution. Our finalized implementation ended up using over 90% of the available BRAM:

```
Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-4

 Number of Slices:                     6241  out of  33792    18%
 Number of Slice Flip Flops:           6130  out of  67584     9%
 Number of 4 input LUTs:               9353  out of  67584    13%
    Number used as logic:              6562
    Number used as Shift registers:    2775
    Number used as RAMs:                 16
 Number of IOs:                        576
 Number of bonded IOBs:                363  out of    684    53%
    IOB Flip Flops:                      2
 Number of BRAMs:                      133  out of    144    92%
 Number of MULT18X18s:                  42  out of    144    29%
 Number of GCLKs:                        6  out of     16    37%
 Number of DCMs:                         3  out of     12    25%
```

Figure 23: Device Utilization Summary in the Synthesis Report

Another challenging aspect of the project was syncing VGA timing with VGA pixel output. The pixels were being passed one-by-one through a series of pipelined modules, and while pipelining the modules was not difficult, tracking the pipeline delays that accumulated required more attention than anticipated. To assist in calculating the delays involved in various paths through the image processing subsystems and then determining by how much the VGA timing signals would need to be delayed to be in sync with the output pixels, we parametrized the timing delays. An included parameter file then specified the delays of each pipelined module and calculated total subsystem delays as well as the maximum possible delay. This approach resolved timing and sync issues.

7 Design Decisions

The first main design decision that we took into consideration was whether to use the Nexys4 or the Labkit. We originally wanted to use the Nexys4 because (1) it was a newer FPGA that interfaced with the Vivado Design Suite, (2) it was portable and had many more switches, (3) it had significantly more block RAM, and (4) it could easily interface with an SD card onto which we could save various background images for usage in our system. After about one week into working with the Nexys4, we ran into issues with the DRAM and camera interfaces, and felt that the lack of documentation for both the DRAM and the Nexys4-compatible camera would potentially become an issue. For these reasons, along with the suggestions from the lab staff, we switched over to the Labkit. While it did not have the some of the conveniences offered by the Nexys4, we were able to benefit from the previous documentation, staff code, and general familiarity of the Labkit.

Another aspect of our design that we discussed when defining the scope of our project was the image data transfer from the labkit to the PC. We considered using Ethernet because of the fast transfer speeds it offered but also understood that implementing an Ethernet interface on the FPGA would involve interconnecting various cores, which could result in far too much time debugging than would be necessary for a project where data transfer was not the entire focus. We then settled on transferring data over UART due to its simplicity and ease of implementation. In regards to the transferred data itself, we discussed sending data in compressed (JPEG) or uncompressed bitmap format; after researching JPEG compression, we realized that the compression algorithm was quite complex and that trying to implement it on the FPGA along with all of our other goals was simply too ambitious. We reasoned that for demonstrating the functionality of our project, transferring the image as a simple uncompressed bitmap file would be enough.

We also initially planned to have custom text be sent over UART, but we decided that it would instead be much more user friendly to use a PS/2 keyboard directly connected to the FPGA. As opposed to the user having to run more code on their PC machine and then wait for text characters to be transferred to the FPGA, they could simply type out what

they wanted onto the keyboard and hit a designated button on the Labkit. In addition, we felt that this added another layer of complexity (and perhaps excitement) to our project by introducing yet another form of user input.

8 Reflections

Lorenzo's Experience

This has been a wonderful experience for me. Though there was a ton of hard work and a lot hours put it, I feel that every second was worth it. It was great to see the final project working and to see the background images load on the screen. It was frustrating at times dealing with the complicated nature of some of the memory systems, such as timing and forecasting with ZBT and Flash in general, but I learned a lot in the process, and now that I am done and looking back, I am very happy that I worked with them. I developed important skills for interfacing with complicated system that are best abstracted to black boxes. For example, after understanding the complicated state machine for Flash, it was only relevant that I manipulate and understand a small number of the total signals. I wish that I had made certain assumptions and realizations earlier, like the bug in the Flash code, but overall I am very happy with how the project when and I am proud of how much I learned and accomplished.

Diana's Experience

I found this project to be an enjoyable and very educational experience. Coming into the class, I wanted to gain experience in designing digital systems with a video or image input; this factored greatly in me wanting to work on the image processing subsystem.

While working on this project, I gained skills in modularizing and pipelining designs, as well as recognizing where a modular approach is needed or when pipelining is necessary. I also realized how important it is to both comment Verilog implementations of pipelined systems and keep track of how pipelining affects timing/syncing of output signals. These skills proved critical to designing and debugging the image processing subsystem.

On a more specific note, I learned about color spaces and the various conversions between them, as well as the attributes that make each of them more suitable for different applications. This project also introduced me to 2D convolution and two filters widely

used in image processing: Gaussian blur and the Sobel operator. Implementing those on the FPGA and seeing them in action was a particularly rewarding experience.

Due to the difficulty of simulating camera input (and visualising it in simulation), most of my debugging was done through generating bitfiles and observing the VGA display during system operation. I did, however, have a chance to use ModelSim for simulating several of the filter and color conversion modules. I also became more familiar with the ISE software and the Core Generator. In retrospect, I wish I had also gained some experience using the logic analyzer, given how useful it can be in debugging.

Overall, this project gave me valuable hands-on experience in building and integrating a complex small-scale digital system with various subsystems. I am very pleased with how the project turned out and with the technical goals that it achieved. I am proud to have participated and contributed to its success.

9 Conclusion

Overall this has been a very successful project. In addition to completing all of out baseline and expected goals to obtain a functioning project, we were able to complete two of our stretch goals.

There we three main things that contributed to our success on the project. First was the very modular approach to our system design. As opposed to having a lot of logic in the top file, we created modules whenever possible and instantiated them in the main file. Second was our clear divide between tasks. By very explicitly separating the parts of this project, for example, image processing vs sprite generation; transfer from FPGA to PC vs. transfer from PC to FPGA; internal memory vs external memory, we were able to divide and conquer. We each became an expert on what we did and then brought the other person up to speed when necessary. That last and most important thing that contributed to our success was the large amounts of time we spent in lab. Whenever possible, we were in lab. We started early and we worked hard the entire project period. Without putting in the time that we did, it would have been impossible to complete this project.

For those considering projects like this one, or for any student in future semesters of 6.111, we recommend heavily that you start early. A project falls behind *"one day at a time"* and starting early gives you a better idea of how long the project may actually take. Project ideas can sometimes appear simple on paper, but they may turn out to be more complex once implementation begins. In addition to providing yourself extra time if things go wrong, which they occasionally do, starting early also can potentially give you additional time at the end to implement stretch goals or simply just have some time to play with the project that you worked so hard to create.

Another tip that we highly recommend is to draw a very detailed block diagram. While block diagrams are very useful in explaining to others how your project works, it is even more beneficial to you. By actually laying out in explicit detail how your design works, you are better able the spot potential problems and understand the system that you are trying to create.

Lastly, don't hesitate to make major changes early on. This was something that we had to do when we switched from the Nexys4 to the Labkit. Once the decision is made, however, stick with it. Don't continue going back and forth because time will be wasted that could have instead been spent on making progress.

10 Acknowledgements

In addition the the code and resources on the 6.111 website, we would also like to give a big thanks to Professor Joe Steinmeyer. His help with both UART and USB interfacing was invaluable and the progress of our project benefited greatly from his assistance.

We also would like to thank the remainder of the lab staff and Professor Gim Hom for helping us through the various issues we encountered over the course of our project. Thank you so much. And thank you for a wonderful semester!

Appendix I - System Usage

Description of User Inputs

switch[7]	sw_ntsc	displays camera input frame on screen
switch[6]	enhance_en	enables saturation / brightness adjustment
switch[5]	filters_en	enables application of filter effects
switch[4]	text_en	enables text overlay over image
switch[3]	graphics_en	enables graphics overlay over image
switch[2]	move_sw	enables text (low) and graphics (high) movement
switch[1]	custom_text_en	enables custom text input and generation
switch[0]	store_bram	initiates BRAM FSM that stores image copy

BUTTON	fsm_state = SEL_BKGD	fsm_state = COLOR_EDITS	fsm_state = ADD_EDITS
up	adjust thresholds	increase saturation	move text/graphics
down	adjust thresholds	decrease saturation	move text/graphics
left	adjust thresholds	decrease brightness	move text/graphics
right	adjust thresholds	increase brightness	move text/graphics
enter	transition to COLOR_EDITS	transition to ADD_EDITS	process custom text
select0	select Paris bkgd	select sepia filter	select mustache
select1	select Rome bkgd	select negative filter	select sunglasses
select2	select Amazon bkdg	select sketch filter	select safari hat
select3	select London bkgd	select cartoon filter	select crown

Hex Display Information

fsm_state[2:0]		INFORMATION SHOWN ON HEX DISPLAY
FSM_IDLE	F	no other additional information
SEL_BKGD	D	hue, saturation, and value thresholds for chroma-keying
COLORS_EDITS	C	saturation and brightness offsets for image enhancement
ADD_EDITS	A	x- and y- coordinates of top left corner of text & graphics
SAVE_TO_BRAM	B	no other additional information
SEND_TO_PC	E	total number of RGB values transferred to PC

The hex nibbles `A-F` denoting the main FSM state are displayed as the leftmost bits on the hex display (`hex_disp[63:60]`) during system operation.

LED Display Information

The LED display provides information on the BRAM state as well as the 2-bit numerical values corresponding to the user's background, filter, and graphic selections.

Instructions for Using the System

1. Turn off all switches to return the system to IDLE state.
2. Make sure that the NTSC is connected to power as well as the Labkit.
3. Stand in front of a green screen facing the camera. Flip the `sw_ntsc` switch to begin. A static frame from the camera feed will be shown on the VGA display.
4. The system is currently in the SEL_BKGD state. You may now adjust the chroma-key detection thresholds by using the directional buttons (as described in Section 4.5 - Chroma-Key Compositing). You may also select a background using the four selector buttons. Press the `enter` button to transition to the next state.

5. The system is currently in the COLOR_EDITS state. You may now enable image enhancement by flipping the enhance_en switch on; when image enhancement is enabled, you may use the directional buttons to adjust the saturation and brightness of the displayed image. You may also enable filter effects by flipping the filters_en switch on; doing so allows you to choose various filter effects using the selector buttons. You may disable image enhancement and filter effects at any time by flipping their respective switches off. Press the enter button to transition.
6. The system is currently in the ADD_EDITS state. You may now enable text overlay by flipping the text_en switch on; doing so will immediately display the default "GREETINGS FROM ___" message. After enabling text overlay, you are able to move the text around the display as long as the move_en switch is off. You may also write your own custom message; to do so, flip the custom_text_en switch on and type a short message on a PS/2 keyboard that is connected to the Labkit. Press the enter button to have the system process your keyboard input and display it on the screen. You may flip the custom_text_en switch off to return to the default greetings message. In this state, you may also enable graphics overlay by flipping the graphics_en switch on; doing so allows you to display one of four graphics. Graphics can be selected using the selector buttons; they can be moved using the directional buttons when the move_en switched is flipped on. When you are satisfied with your edited image, flip the store_bram switch to save to BRAM.
7. The system is currently in the SAVE_TO_BRAM state. What is being displayed on the screen is the data that has been stored in and is now being read out of BRAM. The image will be in 8-bit color and will look cropped on the bottom. You may now begin to transfer this image to a PC. To do so, save the save_bmp.sh, uart_rx.py, and bin2bmp.c files in the same directory. Run save_bmp.sh in a Terminal and wait until "Ready to receive data…" is displayed; then press the enter button on the Labkit to initiate data transfer.
8. The system is currently in the SEND_TO_PC state. After all image data has been transferred, a bitmap file will be generated and saved as pic_out.bmp in the same directory on the PC as the shell script that you just ran.

Appendix II - PC Code

- JPEG to COE Conversion in MATLAB (`JPEGtoCOE.m`) - Lorenzo
- Python COE Parser (`converter.py`) - Lorenzo
- Python UART/USB TX [output to FTDI chip] (`serial1.py`) - Lorenzo
- Python UART/USB RX [input from Teensy] (`uart_rx.py`) - Diana
- Bitmap Generation in C (`bin2bmp.c`) - Diana
- Shell Script (`save_bmp.sh`) - Diana

```
%function img2 = IMG2coe8(imgfile, outfile)
% imgfile = input .jpg file
% outfile = output .coe file
% Example:
%http://www.lbebooks.com/downloads/exportal/vhdl_nexys_example24.pdf
% img2 = IMG2coe8('paris240x160.jpg', 'paris240x160.coe');
img = imread('paris640x480.jpg');
height = size(img, 1);
width = size(img, 2);
s = fopen('paris.coe','wb');  %opens the output file
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
cnt = 0;
%img2 = img;
for r=1:height
    for c=1:width
        cnt = cnt + 1;
        R = img(r,c,1);
        G = img(r,c,2);
        B = img(r,c,3);
        Rb = dec2bin(R,8);
        Gb = dec2bin(G,8);
        Bb = dec2bin(B,8);
%         img2(r,c,1) = bin2dec([Rb(1:5) '000']);
%         img2(r,c,2) = bin2dec([Gb(1:6) '00']);
%         img2(r,c,3) = bin2dec([Bb(1:5) '000']);
        Outbyte = [Rb(1:5) Gb(1:6) Bb(1:5)];
        fprintf(s,Outbyte);

%         if (Outbyte(1:4) == '0000')
%             fprintf(s,'0%X',bin2dec(Outbyte));
%         else
%             fprintf(s,'%X',bin2dec(Outbyte));
%         end
%         if ((c == width) && (r == height))
%             fprintf(s,'%c',';');
%         else
%             if (mod(cnt,32) == 0)
%                 fprintf(s,'%c\n',',');
%             else
%                 fprintf(s,'%c',',');
%             end
%         end
    end
end
fclose(s);
```

```
f = open('london.coe', 'r')
y =open('londond_hex.txt','w')
final=''
for line in f:
        newone=''
        counter=0
        sub=''
        inter=0
        inter2=0
        for letter in line:
                #print letter
                counter+=1
                sub+=letter
                if counter==4:
                        #print sub
                        if sub=='0000':
                                sub="0"
                        if sub=='0001':
                                sub="1"
                        if sub=='0010':
                                sub="2"
                        if sub=='0011':
                                sub="3"
                        if sub=='0100':
                                sub="4"
                        if sub=='0101':
                                sub="5"
                        if sub=='0110':
                                sub="6"
                        if sub=='0111':
                                sub="7"
                        if sub=='1000':
                                sub="8"
                        if sub=='1001':
                                sub="9"
                        if sub=='1010':
                                sub="A"
                        if sub=='1011':
                                sub="B"
                        if sub=='1100':
                                sub="C"
                        if sub=='1101':
                                sub="D"
                        if sub=='1110':
                                sub="E"
                        if sub=='1111':
                                sub="F"
                        #print sub
                        inter+=1
                        if inter==2:
                                sub=sub+' '
                                inter=0
                        inter2+=1
                        if inter2==4:
                                inter2=0
                                sub=sub+'\n'
                        counter=0
                        final+=sub
                        sub=''
y.write(final)
```

```
import serial
import sys
import glob

def serial_ports():
    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in list(range(256))]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this excludes your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')
    print(ports)
    result = []
    return ports
#------------------

ports = serial_ports() #generate list of currently connected serial ports
print (ports)
print("Might need to change the number in code based on what gets printed out")
ser = ports[1]
s = serial.Serial(ser)
print(s)


#f = open('./test.bit','rb')
#command=b"\x39"
f=open('londond_hex.txt','r')
for byteinline in f:
    command = byteinline
    command=bytearray.fromhex(command)
    s.write(command)

s.close()
```

```
#!/usr/bin/python

# 6.111 Final Project - FPGA Passport
# Diana Wofk | MIT 6.111 | Fall 2016

import serial, time
from Queue import *

# Description: serial RX & write to output file
# Input: 640x400x3 bytes, corresponding to 24-bit RGB 640x400 image
# Output: binary file storing the received RGB values


# PARAMETERS
ser = serial.Serial()
#ser.port = "/dev/ttyUSB0"
#ser.port = "/dev/ttyUSB7"
#ser.port = "/dev/ttyS2"
ser.port = "/dev/cu.usbserial-A104VDBZ"
ser.baudrate = 115200
ser.bytesize = serial.EIGHTBITS             #number of bits per bytes
ser.parity = serial.PARITY_NONE             #set parity check: no parity
ser.stopbits = serial.STOPBITS_TWO          #number of stop bits
ser.timeout = None                          #block read
#ser.timeout = 3                            #non-block read
#ser.timeout = 2                            #timeout block read
ser.xonxoff = False                         #disable software flow control
ser.rtscts = False                          #disable hardware (RTS/CTS) flow control
ser.dsrdtr = False                          #disable hardware (DSR/DTR) flow control
ser.writeTimeout = 2                        #timeout for write

#    possible timeout values:
#    1. None: wait forever, block call
#    2. 0: non-blocking mode, return immediately
#    3. x, x is bigger than 0, float allowed, timeout block call


byte_fifo = Queue(maxsize=0)

try:
    ser.open()
except Exception, e:
    print "Error opening serial port: " + str(e)
    exit()

if ser.isOpen():

    print("Ready to receive data...")
    byte_counter = 0

    # enqueue 640x400x3 bytes into fifo
    while byte_counter < 768000:
       byte = ser.read()
       byte_fifo.put(byte)
       byte_counter = byte_counter + 1
       print(byte_counter)

    # dequeue from fifo and write to output file
    f_out = open('data.bin', 'wb')
    while not byte_fifo.empty():
        f_out.write(byte_fifo.get())
    f_out.close()
    exit()

else:
    print "ERROR: cannot open serial port"
```

```
/*****************************************************************************
    bmp.c - read and write bmp images.
    Distributed with Xplanet.
    Copyright (C) 2002 Hari Nair <hari@alumni.caltech.edu>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct BMPHeader
{
    char bfType[2];       /* "BM" */
    int bfSize;           /* Size of file in bytes */
    int bfReserved;       /* set to 0 */
    int bfOffBits;        /* Byte offset to actual bitmap data (= 54) */
    int biSize;           /* Size of BITMAPINFOHEADER, in bytes (= 40) */
    int biWidth;          /* Width of image, in pixels */
    int biHeight;         /* Height of images, in pixels */
    short biPlanes;       /* Number of planes in target device (set to 1) */
    short biBitCount;     /* Bits per pixel (24 in this case) */
    int biCompression;    /* Type of compression (0 if no compression) */
    int biSizeImage;      /* Image size, in bytes (0 if no compression) */
    int biXPelsPerMeter;  /* Resolution in pixels/meter of display device */
    int biYPelsPerMeter;  /* Resolution in pixels/meter of display device */
    int biClrUsed;        /* Number of colors in the color table (if 0, use
                             maximum allowed by biBitCount) */
    int biClrImportant;   /* Number of important colors.  If 0, all colors
                             are important */
};

int
read_bmp(const char *filename, int *width, int *height, unsigned char *rgb)
{
    fprintf(stderr, "Sorry, reading of .bmp files isn't supported yet.\n");
    return(0);
}

int
write_bmp(const char *filename, int width, int height, char *rgb)
{
    int i, j, ipos;
    int bytesPerLine;
    unsigned char *line;

    FILE *file;
    struct BMPHeader bmph;

    /* The length of each line must be a multiple of 4 bytes */

    bytesPerLine = (3 * (width + 1) / 4) * 4;

    strcpy(bmph.bfType, "BM");
    bmph.bfOffBits = 54;
    bmph.bfSize = bmph.bfOffBits + bytesPerLine * height;
    bmph.bfReserved = 0;
    bmph.biSize = 40;
    bmph.biWidth = width;
    bmph.biHeight = height;
```

```
    bmph.biPlanes = 1;
    bmph.biBitCount = 24;
    bmph.biCompression = 0;
    bmph.biSizeImage = bytesPerLine * height;
    bmph.biXPelsPerMeter = 0;
    bmph.biYPelsPerMeter = 0;
    bmph.biClrUsed = 0;
    bmph.biClrImportant = 0;

    file = fopen (filename, "wb");
    if (file == NULL) return(0);

    fwrite(&bmph.bfType, 2, 1, file);
    fwrite(&bmph.bfSize, 4, 1, file);
    fwrite(&bmph.bfReserved, 4, 1, file);
    fwrite(&bmph.bfOffBits, 4, 1, file);
    fwrite(&bmph.biSize, 4, 1, file);
    fwrite(&bmph.biWidth, 4, 1, file);
    fwrite(&bmph.biHeight, 4, 1, file);
    fwrite(&bmph.biPlanes, 2, 1, file);
    fwrite(&bmph.biBitCount, 2, 1, file);
    fwrite(&bmph.biCompression, 4, 1, file);
    fwrite(&bmph.biSizeImage, 4, 1, file);
    fwrite(&bmph.biXPelsPerMeter, 4, 1, file);
    fwrite(&bmph.biYPelsPerMeter, 4, 1, file);
    fwrite(&bmph.biClrUsed, 4, 1, file);
    fwrite(&bmph.biClrImportant, 4, 1, file);

    line = malloc(bytesPerLine);
    if (line == NULL)
    {
        fprintf(stderr, "Can't allocate memory for BMP file.\n");
        return(0);
    }

    for (i = height - 1; i >= 0; i--)
    {
        for (j = 0; j < width; j++)
        {
            ipos = 3 * (width * i + j);
            line[3*j] = rgb[ipos + 2];
            line[3*j+1] = rgb[ipos + 1];
            line[3*j+2] = rgb[ipos];
        }
        fwrite(line, bytesPerLine, 1, file);
    }

    free(line);
    fclose(file);

    return(1);
}

/*****************************************************************************/

/* 6.111 Final Project - FPGA Passport */
/* Diana Wofk | MIT 6.111 | Fall 2016 */

/* Reads input file and saves contents as a string */
/* Ouputs data in stored string as a bitmap file */

int main ()
{
  FILE * fp;
  char * source;

  fp = fopen("data.bin", "rb");
  source = NULL;

  if (fp != NULL)
  {
      /* Go to end of file */
      if (fseek(fp, 0, SEEK_END) == 0)
```

```
      {
          /* Determine size of file buffer */
          long bufsize = ftell(fp);
          if (bufsize > 0) printf("Input file size: %ld bytes\n", bufsize);

          /* Allocate memory for buffer */
          source = malloc(sizeof(char) * (bufsize + 1));

          /* Go to beginning of file */
          if (fseek(fp, 0, SEEK_SET) == 0)
          {
            /* Read file into buffer */
            size_t num_elem = fread(source, sizeof(char), bufsize, fp);
            if (num_elem > 0) {
              printf("Number of elements read: %zd\n", num_elem);
              source[++num_elem] = '\0'; /* Terminate string */
            }
          }
      }
  }

  fclose(fp); /* Close file */

  const char * filename;
  int bmp_width;
  int bmp_height;

  filename = "pic_out.bmp"; /* 640x400 bmp image */
  bmp_width = 640;
  bmp_height = 400;

  write_bmp(filename, bmp_width, bmp_height, source);
  printf("Bitmap file saved. Exiting...\n");

  free(source); /* Free allocated memory */

  return(0);
}
```

```
#!/bin/bash

echo "Connecting..."
python uart_rx.py

gcc -o bin2bmp bin2bmp.c
echo "Creating bitmap..."
./bin2bmp
```

Appendix III - Verilog Source Files

- bram_ifc.v
- cartoon.v
- chroma_key.v
- custom_text.v
- debounce.v
- display_16hex.v
- edge_det.v
- enhance.v
- filters.v
- flash_int.v
- flash_manager.v
- flashreader.v
- flash_write.v
- gaussian.v
- graphics_gen.v
- grayscale.v
- hsv2rgb.v
- invert.v
- line_buf.v
- main_fsm.v
- mover.v
- ntsc2zbt.v
- param.v
- pixel_out.v
- ps2_kbd.v
- ramclock.v
- rgb2hsv.v
- sepia.v
- sepia_tb.v
- sobel_op.v
- test_fsm.v
- text_gen.v
- uart_tx.v
- usb_input.v
- video_decoder.v
- vram_display.v
- vram_flash_img.v
- xvga.v
- ycrcb2rgb.v
- zbt_6111.v
- zbt_6111_sample.v
- labkit.ucf

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:
// Design Name:
// Module Name:    frame_bram_ifc
// Project Name:
// Target Devices:
// Tool versions:
// Description: Module interfacing with BRAM IP. Stores what is currently being
//              displayed via VGA into BRAM; can only store 600*400 bytes. Only
//              600x400 8-bit RGB values are saved of the 640x480 24-bit image.
//              A user input switch enables writing to BRAM; data is read out
//              automatically after it a frame/image has been fully written.
//              Turning switch off resets BRAM FSM and allows stored data to be
//              over-written with new data the next time the switch is raised.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module frame_bram_ifc(
    input clk, rst,
    input store_bram,
    input [10:0] hcount,
    input [9:0] vcount,
    input [10:0] hoffset,
    input [9:0] voffset,
    input [17:0] tx_counter, // read address when transmitting stored data to PC
    input in_display,
    input [23:0] pixel_out,  // pixel displayed via VGA
    input [2:0] fsm_state,
    output [7:0] bram_dout,
    output [1:0] bram_state
  );

  `include "param.v"

  reg [1:0] state_q = BRAM_IDLE;
  assign bram_state = state_q;

  // edge detection on store_bram switch
  reg store_bram_d_q; // delayed store_bram signal
  always @(posedge clk) store_bram_d_q <= store_bram;

  wire bram_sw_rising = !store_bram_d_q && store_bram;
  wire bram_sw_falling = store_bram_d_q && !store_bram;

  // read and write addresses
  reg [17:0] write_counter_q = 0;
  reg [17:0] read_counter_q = 0;

  // control signals for progressing through BRAM FSM states
  wire frame_loaded = (write_counter_q == 18'd255999);      // 600*400 = 256000
  wire at_origin = (hcount==hoffset) && (vcount==voffset);

  // BRAM signal declarations
  wire [7:0] frame_bram_din;
  wire [17:0] frame_bram_addr;
  wire frame_bram_wea;

  bram_ip frame_bram(clk, frame_bram_din, frame_bram_addr, frame_bram_wea, bram_dout
);

  // inputs to BRAM instantiation
  assign frame_bram_din = {pixel_out[23:21],pixel_out[15:13],pixel_out[7:6]};
    // 8-bit color
```

```
  assign frame_bram_wea = (state_q == WRITING_FRAME) && in_display && !frame_loaded;

  assign frame_bram_addr = (fsm_state == SEND_TO_PC) ? tx_counter :
                           (state_q == WRITING_FRAME) ? write_counter_q :
                           (state_q == READING_FRAME) ? read_counter_q : 0;

  always @(posedge clk) begin
    case (state_q)
      BRAM_IDLE     : begin
                        write_counter_q <= 0;
                        read_counter_q <= 0;
                        state_q <= (bram_sw_rising) ? CAPTURE_FRAME : BRAM_IDLE;
                       end

      CAPTURE_FRAME : state_q <= (at_origin) ? WRITING_FRAME : CAPTURE_FRAME;
                      // waiting to begin writing frame into BRAM; begin writing whe
n h&vcount match
                      // offsets, i.e. when pixel_out corresponds to the first (top
left) pixel of frame

      WRITING_FRAME : begin
                        if (frame_bram_wea) write_counter_q <= write_counter_q+1'b1;
                        state_q <= (frame_loaded) ? READING_FRAME : WRITING_FRAME;
                      end

      READING_FRAME : begin
                        // continuously read out from BRAM and output stored data wh
en in the 640x480 display area
                        if (in_display) read_counter_q <= (read_counter_q == 18'd255
999) ? 0 : read_counter_q+1'b1;
                        state_q <= (bram_sw_falling) ? BRAM_IDLE : READING_FRAME;
                      end
    endcase
  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    15:38:38 12/02/2016
// Design Name:
// Module Name:    cartoon_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements a sketch effect as well as a cartoon effect. Takes both
//              24-bit RGB color and grayscale as input. Processes grayscale
//              input with 3x3 Sobel operator to detect edges. Applies 3x3 Gaussian
//              kernel to the color RGB input to create a slight blur. Edge
//              detection produces a sketch effect on its own and a cartoon effect
//              when combined with blurred 8-bit color. 24-bit RGB pixel output.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module cartoon(
    input clk, rst,
    input [10:0] hcount,
    input [9:0] vcount,
    input [7:0] rgb_gray,
    input [23:0] rgb_in,
    output [23:0] rgb_edge,     // sketch filter effect
    output [23:0] rgb_cartoon   // cartoon filter effect
  );

  `include "param.v"

  //////////////////////////////////////////////////////////////////////////////////
  // SOBEL EDGE DETECTION
  //////////////////////////////////////////////////////////////////////////////////

  wire [7:0] a0, a1, a2, a7, pix, a3, a6, a5, a4;
  line_buf #(8) sobel_line_buf(clk, rst, rgb_gray, a0, a1, a2, a7, pix, a3, a6, a5,
a4);

  wire [15:0] gradient;
  sobel_op sobel_op1(clk, rst, a0, a1, a2, a7, a3, a6, a5, a4, gradient);

  wire pixel_edge, cartoon_edge;
  edge_det edge_det1(clk, rst, gradient, pixel_edge, cartoon_edge);

  //////////////////////////////////////////////////////////////////////////////////
  // GAUSSIAN BLUR
  //////////////////////////////////////////////////////////////////////////////////

  wire [2:0] a0r, a1r, a2r, a7r, pix_r, a3r, a6r, a5r, a4r;
  line_buf #(3) gauss_r_buf(clk, rst, rgb_in[23:21], a0r, a1r, a2r, a7r, pix_r, a3r,
 a6r, a5r, a4r);

  wire [2:0] a0g, a1g, a2g, a7g, pix_g, a3g, a6g, a5g, a4g;
  line_buf #(3) gauss_g_buf(clk, rst, rgb_in[15:13], a0g, a1g, a2g, a7g, pix_g, a3g,
 a6g, a5g, a4g);

  wire [1:0] a0b, a1b, a2b, a7b, pix_b, a3b, a6b, a5b, a4b;
  line_buf #(2) gauss_b_buf(clk, rst, rgb_in[7:6], a0b, a1b, a2b, a7b, pix_b, a3b, a
6b, a5b, a4b);

  wire [7:0] rgb_gauss; // 8-bit blurred RGB value
  gaussian gauss_blur(clk, rst, a0r, a1r, a2r, a7r, pix_r, a3r, a6r, a5r, a4r,
                      a0g, a1g, a2g, a7g, pix_g, a3g, a6g, a5g, a4g,
                      a0b, a1b, a2b, a7b, pix_b, a3b, a6b, a5b, a4b, rgb_gauss);

  //////////////////////////////////////////////////////////////////////////////////
```

```
  // RGB OUTPUT
  ////////////////////////////////////////////////////////////////////////////////

  // Blurred color pixels need to be delayed to be in sync with edge detector module
...

  localparam CARTOON_RGB_DLY = SOBEL_DLY-LINE_BUF_DLY-GAUSSIAN_DLY;
  reg [7:0] rgb_color_reg[CARTOON_RGB_DLY-1:0]; // shift register for delaying blurr
ed pixels

  integer i;

  // delay blurred RGB to sync with edge pixel output
  always @(posedge clk) begin
    rgb_color_reg[0] <= rgb_gauss;
    for (i=1; i<(CARTOON_RGB_DLY); i=i+1)
      rgb_color_reg[i] <= rgb_color_reg[i-1];
  end

  wire [7:0] rgb_color = rgb_color_reg[CARTOON_RGB_DLY-1]; // delayed blurred RGB va
lue
  wire [23:0] rgb_color_24bit = {rgb_color[7:5],5'd0,rgb_color[4:2],5'd0,rgb_color[1
:0],6'd0};

  assign rgb_edge = pixel_edge ? 24'h000000 : 24'hFFFFFF;
  assign rgb_cartoon = cartoon_edge ? 24'h000000 : rgb_color_24bit;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    15:30:49 11/22/2016
// Design Name:
// Module Name:    chroma_key
// Project Name:
// Target Devices:
// Tool versions:
// Description: Determines whether the HSV values of the input pixel fall into a
//              pre-defined range and if so, marks that pixel as having matched
//              the chroma key. Nominal HSV threshholds are defined as parameters,
//              but threshholds used in color detection can be adjusted by user.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module chroma_key(
    input clk, rst,
    input vsync,
    input [23:0] hsv_chr_in,
    input up, down, left, right,
    input adjust_thr_en,  // allows user to adjust threshholds
    output [7:0] range, h_nom, s_nom, v_nom,
    output reg [23:0] hsv_chr_out,
    output reg chroma_key_match
  );

  localparam H_NOMINAL = 8'd85;
  //localparam H_RANGE_POS = 8'd50;
  //localparam H_RANGE_NEG = 8'd50;

  localparam S_NOMINAL = 8'd94;
  //localparam S_RANGE_POS = 8'd50;
  //localparam S_RANGE_NEG = 8'd50;

  localparam V_NOMINAL = 8'd202;
  localparam V_RANGE_POS = 8'd50;
  localparam V_RANGE_NEG = 8'd100;

  localparam RANGE_NOMINAL = 8'd50;

  // HSV threshholds that can all be adjusted by user
  reg [7:0] h_nom_q = H_NOMINAL;
  reg [7:0] s_nom_q = S_NOMINAL;
  reg [7:0] v_nom_q = V_NOMINAL;
  reg [7:0] range_q = RANGE_NOMINAL; // threshhold +/- range for H and S

  assign {range, h_nom, s_nom, v_nom} = {range_q, h_nom_q, s_nom_q, v_nom_q};

  // max and min for H
  wire [7:0] h_max = h_nom_q+range_q;
  wire [7:0] h_min = h_nom_q-range_q;

  // max and min for S
  wire [7:0] s_max = s_nom_q+range_q;
  wire [7:0] s_min = s_nom_q-range_q;

  // max and min for V
  wire [7:0] v_max = v_nom_q+V_RANGE_POS;
  wire [7:0] v_min = v_nom_q-V_RANGE_NEG;

  // HSV values of input pixel
  wire [7:0] h_chr_in = hsv_chr_in[23:16];
  wire [7:0] s_chr_in = hsv_chr_in[15:8];
  wire [7:0] v_chr_in = hsv_chr_in[7:0];
```

```
  // determine whether HSV values are in range
  wire h_in_range = (h_chr_in >= h_min) && (h_chr_in <= h_max);
  wire s_in_range = (s_chr_in >= s_min) && (s_chr_in <= s_max);
  wire v_in_range = (v_chr_in >= v_min) && (v_chr_in <= v_max);
  wire in_range = h_in_range && s_in_range && v_in_range;

  // detect falling edge of vsync
  reg vsync_q;
  always @(posedge clk) vsync_q <= vsync;

  wire vsync_falling;
  assign vsync_falling = (vsync !== vsync_q) && (vsync === 1'b0);

  reg [3:0] vsync_counter = 0;

  always @(posedge clk) begin
    hsv_chr_out <= hsv_chr_in;
    chroma_key_match <= (in_range) ? 1 : 0;

    if (rst) begin
      h_nom_q <= H_NOMINAL;
      s_nom_q <= S_NOMINAL;
      v_nom_q <= V_NOMINAL;
      range_q <= RANGE_NOMINAL;
    end else if (vsync_falling && adjust_thr_en) begin
      vsync_counter <= vsync_counter+1'b1; // to slow down adjustment speed
      if (vsync_counter == 4'd15) begin
        if (!left && !right) begin              // change H threshold
          if (up && (h_nom_q < 8'd255)) h_nom_q <= h_nom_q+1'b1;
          if (down && (h_nom_q > 8'd0)) h_nom_q <= h_nom_q-1'b1;
        end else if (left && !right) begin      // change S threshold
          if (up && (s_nom_q < 8'd255)) s_nom_q <= s_nom_q+1'b1;
          if (down && (s_nom_q > 8'd0)) s_nom_q <= s_nom_q-1'b1;
        end else if (!left && right) begin      // change V threshold
          if (up && (v_nom_q < 8'd255)) v_nom_q <= v_nom_q+1'b1;
          if (down && (v_nom_q > 8'd0)) v_nom_q <= v_nom_q-1'b1;
        end else if (left && right) begin      // change threshold range
          if (up && (range_q < 8'd255)) range_q <= range_q+1'b1;
          if (down && (range_q > 8'd0)) range_q <= range_q-1'b1;
        end
      end
    end
  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    17:44:19 11/30/2016
// Design Name:
// Module Name:    custom_text
// Project Name:
// Target Devices:
// Tool versions:
// Description: When custom text is enabled, this module takes in keyboard data
//              from ps2_ascii_input and outputs a string containing the user-
//              entered characters. Default maximum string length is 20.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module custom_text #(parameter TEXT_LEN_MAX=20) (
    input clock_27mhz, reset,
    input custom_text_en,
    input [2:0] fsm_state,
    input button_enter,       // "done" signal
    // Keyboard Signals
    input keyboard_clock,
    input keyboard_data,
    // Outputs
    output [TEXT_LEN_MAX*8-1:0] char_array,
    output char_array_rdy,
    output [5:0] num_char
  );

  `include "param.v"

  // Keyboard Signals
  wire [7:0] ascii;
  wire char_rdy;

  ps2_ascii_input keyboard(
    .clock_27mhz      (clock_27mhz),
    .reset            (reset),
    .clock            (keyboard_clock),
              .data             (keyboard_data),
    .ascii            (ascii),
    .ascii_ready      (char_rdy)
  );

  reg [TEXT_LEN_MAX*8-1:0] char_array_q = 0;

  // states for keyboard input
  reg accepting_kbd_input_q = 1'b0;
  reg processing_input_q = 1'b0;

  reg [5:0] char_ctr_q = TEXT_LEN_MAX;  // # of available char left
  reg [5:0] num_char_q = 0;             // # of char user has entered

  // enable keyboard input processing when custom_text_en is asserted
  wire kbd_input_en = (fsm_state == ADD_EDITS) && custom_text_en;
  // clear char array and reset counter when custom_text_en is deasserted
  wire reset_kbd_input = (fsm_state == ADD_EDITS) && !custom_text_en;

  //output assignments
  assign char_array = char_array_q;
  assign char_array_rdy = processing_input_q;
  assign num_char = num_char_q;

  integer i;

  always @(posedge clock_27mhz) begin
```

```
    if (reset_kbd_input) begin
      accepting_kbd_input_q <= 1'b0;
      processing_input_q <= 1'b0;
      char_array_q <= 0;
      char_ctr_q <= TEXT_LEN_MAX;
    end else if (kbd_input_en) begin
      if (button_enter) begin
        // user input is done; enable ready signal
        accepting_kbd_input_q <= 1'b0;
        processing_input_q <= 1'b1;      // connected to char array rdy
        num_char_q <= TEXT_LEN_MAX-char_ctr_q;
      end else accepting_kbd_input_q <= ~processing_input_q;
      if (accepting_kbd_input_q && char_rdy && (char_ctr_q > 0)) begin
        // new char byte has been entered by the user
        // store new byte in array and decrement char counter
        char_ctr_q <= char_ctr_q - 1'b1;
        // this array is flipped, meaning that the first user character
        // entered will be written as leftmost byte in array
        for (i=TEXT_LEN_MAX; i>0; i=i-1)
           if (char_ctr_q==i) char_array_q[(8*(i-1))+:8] <= ascii;
      end
    end
  end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module
//
////////////////////////////////////////////////////////////////////////////////

module debounce (reset, clk, noisy, clean);
   input reset, clk, noisy;
   output clean;

   parameter NDELAY = 650000;
   parameter NBITS = 20;

   reg [NBITS-1:0] count;
   reg xnew, clean;

   always @(posedge clk)
     if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
     else if (count == NDELAY) clean <= xnew;
     else count <= count+1;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
////////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
                      disp_blank, disp_clock, disp_rs, disp_ce_b,
                      disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data_in;                // 16 hex nibbles to display

   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
          disp_reset_b;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   ////////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   ////////////////////////////////////////////////////////////////////////////

   reg [5:0] count;
   reg [7:0] reset_count;
//   reg       old_clock;
   wire      dreset;
   wire      clock = (count<27) ? 0 : 1;

   always @(posedge clock_27mhz)
     begin
        count <= reset ? 0 : (count==53 ? 0 : count+1);
        reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//      old_clock <= clock;
     end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

   ////////////////////////////////////////////////////////////////////////////
   //
   // Display State Machine
   //
   ////////////////////////////////////////////////////////////////////////////
```

```
   reg [7:0] state;             // FSM state
   reg [9:0] dot_index;         // index to current dot being clocked out
   reg [31:0] control;          // control register
   reg [3:0] char_index;        // index of current character
   reg [39:0] dots;             // dots for a single digit
   reg [3:0] nibble;            // hex nibble of current character
   reg [63:0] data;

   assign disp_blank = 1'b0; // low <= not blanked

   always @(posedge clock_27mhz)
     if (clock_tick)
       begin
          if (dreset)
            begin
               state <= 0;
               dot_index <= 0;
               control <= 32'h7F7F7F7F;
            end
          else
            casex (state)
              8'h00:
                begin
                   // Reset displays
                   disp_data_out <= 1'b0;
                   disp_rs <= 1'b0; // dot register
                   disp_ce_b <= 1'b1;
                   disp_reset_b <= 1'b0;
                   dot_index <= 0;
                   state <= state+1;
                end

              8'h01:
                begin
                   // End reset
                   disp_reset_b <= 1'b1;
                   state <= state+1;
                end

              8'h02:
                begin
                   // Initialize dot register (set all dots to zero)
                   disp_ce_b <= 1'b0;
                   disp_data_out <= 1'b0; // dot_index[0];
                   if (dot_index == 639)
                     state <= state+1;
                   else
                     dot_index <= dot_index+1;
                end

              8'h03:
                begin
                   // Latch dot data
                   disp_ce_b <= 1'b1;
                   dot_index <= 31;              // re-purpose to init ctrl reg
                   state <= state+1;
                end

              8'h04:
                begin
                   // Setup the control register
                   disp_rs <= 1'b1; // Select the control register
                   disp_ce_b <= 1'b0;
                   disp_data_out <= control[31];
                   control <= {control[30:0], 1'b0};    // shift left
                   if (dot_index == 0)
                     state <= state+1;
                   else
                     dot_index <= dot_index-1;
                end

              8'h05:
```

```
                begin
                   // Latch the control register data / dot data
                   disp_ce_b <= 1'b1;
                   dot_index <= 39;              // init for single char
                   char_index <= 15;             // start with MS char
                   data <= data_in;
                   state <= state+1;
                end

              8'h06:
                begin
                   // Load the user's dot data into the dot reg, char by char
                   disp_rs <= 1'b0;                      // Select the dot register
                   disp_ce_b <= 1'b0;
                   disp_data_out <= dots[dot_index]; // dot data from msb
                   if (dot_index == 0)
                     if (char_index == 0)
                       state <= 5;                       // all done, latch data
                     else
                       begin
                          char_index <= char_index - 1; // goto next char
                          data <= data_in;
                          dot_index <= 39;
                       end
                   else
                     dot_index <= dot_index-1;  // else loop thru all dots
                end

            endcase // casex(state)
        end

    always @ (data or char_index)
      case (char_index)
        4'h0:            nibble <= data[3:0];
        4'h1:            nibble <= data[7:4];
        4'h2:            nibble <= data[11:8];
        4'h3:            nibble <= data[15:12];
        4'h4:            nibble <= data[19:16];
        4'h5:            nibble <= data[23:20];
        4'h6:            nibble <= data[27:24];
        4'h7:            nibble <= data[31:28];
        4'h8:            nibble <= data[35:32];
        4'h9:            nibble <= data[39:36];
        4'hA:            nibble <= data[43:40];
        4'hB:            nibble <= data[47:44];
        4'hC:            nibble <= data[51:48];
        4'hD:            nibble <= data[55:52];
        4'hE:            nibble <= data[59:56];
        4'hF:            nibble <= data[63:60];
      endcase

    always @(nibble)
      case (nibble)
        4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
        4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
        4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
        4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
        4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
        4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
        4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
        4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
        4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
        4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
        4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
        4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
        4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
        4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
        4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
        4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
      endcase

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    14:54:30 12/02/2016
// Design Name:
// Module Name:    edge_det
// Project Name:
// Target Devices:
// Tool versions:
// Description: Compares input gradient to a pre-defined threshold. If the
//              gradient exceeds the threshold, pixel is marked as an edge;
//              correct operation requires the output of this module to be
//              in sync with the pixel it applies to (the center pixel whose
//              neighboring pixels were processed in the sobel_op module).
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module edge_det(
    input clk, rst,
    input [15:0] gradient,
    output reg pixel_edge,
    output reg cartoon_edge
  );

  `include "param.v"

  always @(posedge clk) begin
    pixel_edge <= (gradient > GRADIENT_EDGE_THRESHOLD);   // for sketch effect
    cartoon_edge <= (gradient > CARTOON_EDGE_THRESHOLD);  // for cartoon effect
  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    11:02:04 11/18/2016
// Design Name:
// Module Name:    enhance
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements the image enhacement feature, which allows users to
//              adjust the saturation and brightness values of pixels in an
//              image and view the changes on screen in real-time.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module enhance(
    input clk, rst,
    input vsync,
    input enhance_en,           // enable image enhancement
    input enhance_user_in_en,   // allow user to adjust S and V values
    input inc_saturation,
    input dec_saturation,
    input inc_brightness,
    input dec_brightness,
    input [23:0] hsv_in,
    output reg [23:0] hsv_out,
    output [7:0] s_offset, v_offset,
    output s_dir, v_dir
  );

  parameter S_DEV = 1; // SATURATION
  parameter V_DEV = 1; // BRIGHTNESS

  reg [7:0] s_offset_q, v_offset_q;
  reg s_dir_q, v_dir_q;

  assign {s_dir, s_offset} = {s_dir_q, s_offset_q};
  assign {v_dir, v_offset} = {v_dir_q, v_offset_q};

  // detect falling edge of vsync
  reg vsync_q;
  always @(posedge clk) vsync_q <= vsync;

  wire vsync_falling;
  assign vsync_falling = (vsync !== vsync_q) && (vsync === 1'b0);

  wire zero_saturation = inc_saturation && dec_saturation; // don't change saturatio
n
  wire zero_brightness = inc_brightness && dec_brightness; // don't change brightnes
s

  // reset: set saturation and brightness to original values
  wire reset_enhance = enhance_user_in_en && zero_saturation && zero_brightness;

  // adjust S/V offsets based on user input
  always @(posedge clk) begin
    if (reset_enhance) begin
      s_offset_q <= 0;
      v_offset_q <= 0;
      s_dir_q <= 0;
      v_dir_q <= 0;
    end else if (vsync_falling && enhance_user_in_en) begin

      // adjust saturation offset
      case ({inc_saturation, dec_saturation, s_dir_q})
        3'b100 : begin
```

```
                    if (s_offset_q < S_DEV) begin
                      s_offset_q <= S_DEV - s_offset_q;
                      s_dir_q <= 1'b1;  // switch to positive offset
                    end else s_offset_q <= s_offset_q - S_DEV;
                  end
          3'b101  : begin
                    if (s_offset_q < (8'd255-S_DEV)) s_offset_q <= s_offset_q + S_DE
V;
                    else s_offset_q <= 8'd255;
                  end
          3'b011  : begin
                    if (s_offset_q < S_DEV) begin
                      s_offset_q <= S_DEV - s_offset_q;
                      s_dir_q <= 1'b0;  // switch to negative offset
                    end else s_offset_q <= s_offset_q - S_DEV;
                  end
          3'b010  : begin
                    if (s_offset_q < (8'd255-S_DEV)) s_offset_q <= s_offset_q + S_DE
V;
                    else s_offset_q <= 8'd255;
                  end
          default : ;   // simultaneous inc/dec produces no change in offset
        endcase

        // adjust brightness offset
        case ({inc_brightness, dec_brightness, v_dir_q})
          3'b100 : begin
                    if (v_offset_q < V_DEV) begin
                      v_offset_q <= V_DEV - v_offset_q;
                      v_dir_q <= 1'b1;  // switch to positive offset
                    end else v_offset_q <= v_offset_q - V_DEV;
                  end
          3'b101  : begin
                    if (v_offset_q < (8'd255-V_DEV)) v_offset_q <= v_offset_q + V_DE
V;
                    else v_offset_q <= 8'd255;
                  end
          3'b011  : begin
                    if (v_offset_q < V_DEV) begin
                      v_offset_q <= V_DEV - v_offset_q;
                      v_dir_q <= 1'b0;  // switch to negative offset
                    end else v_offset_q <= v_offset_q - V_DEV;
                  end
          3'b010  : begin
                    if (v_offset_q < (8'd255-V_DEV)) v_offset_q <= v_offset_q + V_DE
V;
                    else v_offset_q <= 8'd255;
                  end
          default : ;   // simultaneous inc/dec produces no change in offset
        endcase

      end
    end

    // apply S/V offsets to HSV input; latency = 1 clk cycle
    always @(posedge clk) begin
      if (!enhance_en) hsv_out <= hsv_in;
      else begin
        hsv_out[23:16] <= hsv_in[23:16];  // hue stays the same

        // apply saturation offset
        if (s_dir_q) begin
          if (hsv_in[15:8] < (8'd255 - s_offset_q))
            hsv_out[15:8] <= hsv_in[15:8] + s_offset_q;
          else hsv_out[15:8] <= 8'd255;
        end else begin
          if (hsv_in[15:8] > s_offset_q)
            hsv_out[15:8] <= hsv_in[15:8] - s_offset_q;
          else hsv_out[15:8] <= 8'd0;
        end

        // apply brightness offset
        if (v_dir_q) begin
```

```
        if (hsv_in[7:0] < (8'd255 - v_offset_q))
          hsv_out[7:0] <= hsv_in[7:0] + v_offset_q;
        else hsv_out[7:0] <= 8'd255;
      end else begin
        if (hsv_in[7:0] > v_offset_q)
          hsv_out[7:0] <= hsv_in[7:0] - v_offset_q;
        else hsv_out[7:0] <= 8'd0;
      end

    end
  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    20:17:14 11/20/2016
// Design Name:
// Module Name:    filters
// Project Name:
// Target Devices:
// Tool versions:
// Description: Instantiates 4 filter effects modules. Processes user
//              button input to determine which filter has been selected.
//              Chooses output pixel based on the selected filter.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module filters(
    input clk, rst,
    input filters_en,           // enable filters
    input filters_user_in_en,   // assert to allow users to change filter
    input select0,
    input select1,
    input select2,
    input select3,
    input [10:0] hcount,
    input [9:0] vcount,
    input [23:0] rgb_in,
    output [23:0] rgb_out,
    output [2:0] filter
  );

  `include "param.v"

  reg [2:0] filter_q = GRAYSCALE;   // default to grayscale

  wire sel_all = select0 && select1 && select2 && select3;  // all buttons pressed

  // determine which filter was selected
  always @(posedge clk) begin
    if (filters_en && filters_user_in_en && select0) filter_q <= SEPIA;
    if (filters_en && filters_user_in_en && select1) filter_q <= INVERT;
    if (filters_en && filters_user_in_en && select2) filter_q <= EDGE;
    if (filters_en && filters_user_in_en && select3) filter_q <= CARTOON;
    if (filters_en && filters_user_in_en && sel_all) filter_q <= GRAYSCALE;
  end

  // Filter Instantiations
  wire [23:0] rgb_invert;
  wire [23:0] rgb_sepia;
  wire [23:0] rgb_gray;
  wire [23:0] rgb_edge;
  wire [23:0] rgb_cartoon;

  sepia sepia_filter(
    .clk    (clk),
    .rst    (rst),
    .r_in   (rgb_in[23:16]),
    .g_in   (rgb_in[15:8]),
    .b_in   (rgb_in[7:0]),
    .r_out  (rgb_sepia[23:16]),
    .g_out  (rgb_sepia[15:8]),
    .b_out  (rgb_sepia[7:0])
  );

  invert invert_filter(
    .clk    (clk),
    .rst    (rst),
```

```
    .r_in   (rgb_in[23:16]),
    .g_in   (rgb_in[15:8]),
    .b_in   (rgb_in[7:0]),
    .r_out  (rgb_invert[23:16]),
    .g_out  (rgb_invert[15:8]),
    .b_out  (rgb_invert[7:0])
  );

  grayscale grayscale_filter(
    .clk    (clk),
    .rst    (rst),
    .r_in   (rgb_in[23:16]),
    .g_in   (rgb_in[15:8]),
    .b_in   (rgb_in[7:0]),
    .r_out  (rgb_gray[23:16]),
    .g_out  (rgb_gray[15:8]),
    .b_out  (rgb_gray[7:0])
  );

  cartoon cartoon_filter(
    .clk          (clk),
    .rst          (rst),
    .hcount       (hcount),
    .vcount       (vcount),
    .rgb_gray     (rgb_gray[7:0]),
    .rgb_in       (rgb_in),
    .rgb_edge     (rgb_edge),
    .rgb_cartoon  (rgb_cartoon)
  );

  // Output Assignments
  assign filter = filter_q;
  assign rgb_out = (!filters_en) ? rgb_in :
                   (filter_q == SEPIA) ? rgb_sepia :
                   (filter_q == INVERT) ? rgb_invert :
                   (filter_q == EDGE) ? rgb_edge :
                   (filter_q == CARTOON) ? rgb_cartoon :
                   (filter_q == GRAYSCALE) ? rgb_gray : rgb_in;

endmodule
```

```
//flash interface
module flash_int(reset, clock, op, address, wdata, rdata, busy, flash_data,
                 flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                 flash_reset_b, flash_sts, flash_byte_b);

   parameter access_cycles = 5;
   parameter reset_assert_cycles = 1000;
   parameter reset_recovery_cycles = 30;

   input reset, clock; // Reset and clock for the flash interface
   input [1:0] op; // Flash operation select (read, write, idle)
   input [22:0] address;
   input [15:0] wdata;
   output [15:0] rdata;
   output busy;
   inout [15:0] flash_data;
   output [23:0] flash_address;
   output flash_ce_b, flash_oe_b, flash_we_b;
   output flash_reset_b, flash_byte_b;
   input  flash_sts;

   reg [1:0] lop;
   reg [15:0] rdata;
   reg busy;
   reg [15:0] flash_wdata;
   reg flash_ddata;
   reg [23:0] flash_address;
   reg flash_oe_b, flash_we_b, flash_reset_b;

   assign flash_ce_b = flash_oe_b && flash_we_b;
   assign flash_byte_b = 1; // 1 = 16-bit mode (A0 ignored)

   assign flash_data = flash_ddata ? flash_wdata : 16'hZ;

   initial
     flash_reset_b <= 1'b1;

   reg [9:0] state;

   always @(posedge clock)
     if (reset)
       begin
                       state <= 0;
                       flash_reset_b <= 0;
                       flash_we_b <= 1;
                       flash_oe_b <= 1;
                       flash_ddata <= 0;
                       busy <= 1;
       end
     else if (flash_reset_b == 0)
       if (state == reset_assert_cycles)
                       begin
                               flash_reset_b <= 1;
                               state <= 1023-reset_recovery_cycles;
                       end
       else
                       state <= state+1;
         else if ((state == 0) && !busy)
       // The flash chip and this state machine are both idle. Latch the user's
       // address and write data inputs. Deassert OE and WE, and stop driving
       // the data buss ourselves. If a flash operation (read or write) is
       // requested, move to the next state.
       begin
                       flash_address <= {address, 1'b0};
                       flash_we_b <= 1;
                       flash_oe_b <= 1;
                       flash_ddata <= 0;
                       flash_wdata <= wdata;
                       lop <= op;
                       if (op != `FLASHOP_IDLE)
                               begin
                                       busy <= 1;
                                       state <= state+1;
```

```
                        end
                    else
                        busy <= 0;
                    end
                else if ((state==0) && flash_sts)
                    busy <= 0;
                else if (state == 1)
      // The first stage of a flash operation. The address bus is already set,
      // so, if this is a read, we assert OE. For a write, we start driving
      // the user's data onto the flash databus (the value was latched in the
      // previous state.
                    begin
                        if (lop == `FLASHOP_WRITE)
                            flash_ddata <= 1;
                        else if (lop == `FLASHOP_READ)
                            flash_oe_b <= 0;
                        state <= state+1;
      end
    else if (state == 2)
      // The second stage of a flash operation. Nothing to do for a read. For
      // a write, we assert WE.
      begin
                    if (lop == `FLASHOP_WRITE)
                        flash_we_b <= 0;
                    state <= state+1;
      end
    else if (state == access_cycles+1)
      // The third stage of a flash operation. For a read, we latch the data
      // from the flash chip. For a write, we deassert WE.
      begin
                    if (lop == `FLASHOP_WRITE)
                        flash_we_b <= 1;
                    if (lop == `FLASHOP_READ)
                        rdata <= flash_data;
                    state <= 0;
      end
    else
      begin
                    if (!flash_sts)
                        busy <= 1;
                    state <= state+1;
      end

endmodule
```

```
//manages all the stuff needed to read and write to the flash ROM
module flash_manager(clock, reset, dots, writemode, wdata, dowrite, raddr, frdata, d
oread, busy, flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b, flash_re
set_b, flash_sts, flash_byte_b, fsmstate);
        input reset, clock;                     //clock and reset
        output [639:0] dots;            //outputs to dot-matrix to help debug flash,
 not necessary
        input writemode;                        //if true then we're in write mode,
else we're in read mode
        input [15:0] wdata;                     //data to be written
        input dowrite;                          //putting this high tells the manage
r the data it has is new, write it
        input [22:0] raddr;                     //address to read from
        output[15:0] frdata;            //data being read
        reg[15:0]    rdata;
        input doread;                           //putting this high tells the manage
r to perform a read on the current address
        output busy;                            //and an output to tell folks we're
still working on the last thing
        reg busy;

        inout [15:0] flash_data;                                        //direct pas
sthrough from labkit to low-level modules (flash_int and test_fsm)
    output [23:0] flash_address;
    output flash_ce_b, flash_oe_b, flash_we_b;
    output flash_reset_b, flash_byte_b;
    input  flash_sts;

        wire flash_busy;                //except these, which are internal to the in
terface
        wire[15:0] fwdata;
        wire[15:0] frdata;
        wire[22:0] address;
        wire [1:0] op;

        reg [1:0] mode;
        wire fsm_busy;

        reg[2:0] state=2;                                       //210

        output[11:0] fsmstate;
        wire [7:0] fsmstateinv;
        assign fsmstate = {state,flash_busy,fsm_busy,fsmstateinv[4:0],mode};    //fo
r debugging only

                                                                                //th
is guy takes care of /some/ of flash's tantrums
        flash_int flash(reset, clock, op, address, fwdata, frdata, flash_busy, flash
_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_sts,
flash_byte_b);
                                                                                //an
d this guy takes care of the rest of its tantrums
        test_fsm  fsm  (reset, clock, op, address, fwdata, frdata, flash_busy, dots,
 mode, fsm_busy, wdata, raddr, fsmstateinv);

        parameter MODE_IDLE     = 0;
        parameter MODE_INIT     = 1;
        parameter MODE_WRITE = 2;
        parameter MODE_READ     = 3;

        parameter HOME          = 3'd0;
        parameter MEM_INIT      = 3'd1;
        parameter MEM_WAIT      = 3'd2;
        parameter WRITE_READY= 3'd3;
        parameter WRITE_WAIT    = 3'd4;
        parameter READ_READY    = 3'd5;
        parameter READ_WAIT     = 3'd6;

        always @ (posedge clock)
                if(reset)
                        begin
                                busy <= 1;
                                state <= HOME;
```

```
                                mode <= MODE_IDLE;
                        end
                else begin
                        case(state)
                                HOME://0                                        //we always
start here
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        if(writemode)
                                                                begin
                                                                        busy <= 1;
                                                                        state <= MEM
_INIT;
                                                                end
                                                        else
                                                                begin
                                                                        busy <= 1;
                                                                        state <= REA
D_READY;
                                                                end
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                MEM_INIT://1
            //begin wiping the memory
                                        begin
                                                busy <= 1;
                                                mode <= MODE_INIT;
                                                if(fsm_busy)
            //to give the fsm a chance to raise its busy signal
                                                        state <= MEM_WAIT;
                                        end

                                MEM_WAIT://2
    //finished wiping
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        state<= WRITE_READY;
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                WRITE_READY://3                                 //wa
iting for data to write to flash
                                        if(dowrite)
                                                begin
                                                        busy <= 1;
                                                        mode <= MODE_WRITE;
                                                end
                                        else if(busy)
                                                state <= WRITE_WAIT;
                                        else if(!writemode)
                                                state <= READ_READY;

                                WRITE_WAIT://4                                  //waiting fo
r flash to finish writing
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        state <= WRITE_READY;
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                READ_READY://5                                  //ready to r
ead data
                                        if(doread)
                                                begin
                                                        busy <= 1;
                                                        mode <= MODE_READ;
```

```
                                        if(busy)
    //lets the fsm raise its busy level
                                                state <= READ_WAIT;
                                        end
                                else
                                        busy <= 0;

                        READ_WAIT://6                   //waiting for flash
to give the data up
                                if(!fsm_busy)
                                        begin
                                                busy <= 0;
                                                state <= READ_READY;
                                        end
                                else
                                        mode <= MODE_IDLE;

                        default: begin          //should never happen...
                                state <= 3'd7;
                        end
                endcase
        end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Lorenzo Vigano
//
// Create Date:    21:00:02 12/07/2016
// Design Name:
// Module Name:    flashreader
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module flashreader(clk,flashreset,busy,start,pictsel, flashaddr,writemode,dowrite,do
read,wdata,loaded);
                input clk;
                input flashreset;
                input busy;
                input start;
                input [2:0] pictsel;
                output reg loaded=0;
                output reg [22:0] flashaddr;
                output reg writemode, dowrite, doread, wdata;

                reg waitone=0;
                reg has=0;
                reg started=0;
                reg [18:0] flashcounter=0;
                always @(posedge clk) begin
                        if (flashreset) begin //shouldn't ever happen
                                writemode <=1;
                                dowrite <= 0;
                                doread <= 0;
                                wdata <= 0; // initial write data = 0
                                flashaddr <= 0; // initial read address = 0
                                end
                        if (start) started<=1; //if given signal to start caching an
 image
                        if (busy==0) begin //only do things when flash isnt busy
                                if(started) begin
                                        loaded<=0;
                                        writemode<=0;
                                        doread<=1;
                                        waitone<=0;
                                        has<=0;
                                        case (pictsel)
                                                0:begin //paris
                                                        flashaddr<=22'd307201;
                                                        flashcounter<=307200; //640*
480
                                                        end
                                                1:begin //rome
                                                        flashaddr<=22'd614401;
                                                        flashcounter<=307200;
                                                        end
                                                2:begin //amazon
                                                        flashaddr<=22'd1152001;
                                                        flashcounter<=307200;
                                                        end
                                                3:begin //london
                                                        flashaddr<=22'd1;
                                                        flashcounter<=307200;
                                                        end
                                                4:begin //start
                                                        flashaddr<=22'd921601;
                                                        flashcounter<=260400;//640*3
```

```
60
                                                end
                                        endcase
                                        started<=0; //done intializing things. now g
o to steady state code
                                end
                                else begin //steady state code
                                        if (flashcounter!=0) begin
                                                flashcounter<=flashcounter-1;
                                                flashaddr<=flashaddr+1;
                                                end
                                        else begin //picture has been sent to zbt
                                                loaded<=1;
                                                end
                                end
                        end
                end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module (video version - 24 bits)
//
//////////////////////////////////////////////////////////////////////////////////

module debounce (input reset, clock, noisy,
                 output reg clean);

   reg [19:0] count;
   reg new;

   always @(posedge clock)
     if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != new) begin new <= noisy; count <= 0; end
     else if (count == 650000) clean <= new;
     else count <= count+1;

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2012-Sep-15: Converted to 24bit RGB
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
```

```
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////////////////////////////////////////////////

module lab3   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
               ac97_bit_clock,

               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
               vga_out_vsync,

               tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

               tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

               clock_feedback_out, clock_feedback_in,

               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
               flash_reset_b, flash_sts, flash_byte_b,

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_in,

               button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up,

               switch,

               led,

               user1, user2, user3, user4,

               daughtercard,

               systemace_data, systemace_address, systemace_ce_b,
               systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

               analyzer1_data, analyzer1_clock,
               analyzer2_data, analyzer2_clock,
               analyzer3_data, analyzer3_clock,
               analyzer4_data, analyzer4_clock);

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
          vga_out_hsync, vga_out_vsync;

   output [9:0] tv_out_ycrcb;
```

```
   output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
          tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
          tv_out_subcar_reset;

   input  [19:0] tv_in_ycrcb;
   input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
          tv_in_hff, tv_in_aff;
   output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
          tv_in_reset_b, tv_in_clock;
   inout  tv_in_i2c_data;

   inout  [35:0] ram0_data;
   output [18:0] ram0_address;
   output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
   output [3:0] ram0_bwe_b;

   inout  [35:0] ram1_data;
   output [18:0] ram1_address;
   output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
   output [3:0] ram1_bwe_b;

   input  clock_feedback_in;
   output clock_feedback_out;

   inout  [15:0] flash_data;
   output [23:0] flash_address;
   output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
   input  flash_sts;

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

   output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
   input  disp_data_in;
   output  disp_data_out;

   input  button0, button1, button2, button3, button_enter, button_right,
          button_left, button_down, button_up;
   input  [7:0] switch;
   output [7:0] led;

   inout [31:0] user1, user2, user3, user4;

   inout [43:0] daughtercard;

   inout  [15:0] systemace_data;
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
                 analyzer4_data;
   output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

   ////////////////////////////////////////////////////////////////////////
   //
   // I/O Assignments
   //
   ////////////////////////////////////////////////////////////////////////

   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   assign ac97_sdata_out = 1'b0;
   // ac97_sdata_in is an input

   // Video Output
   assign tv_out_ycrcb = 10'h0;
```

```
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'b1;
   assign ram0_ce_b = 1'b1;
   assign ram0_oe_b = 1'b1;
   assign ram0_we_b = 1'b1;
   assign ram0_bwe_b = 4'hF;
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
   assign clock_feedback_out = 1'b0;
   // clock_feedback_in is an input

   // Flash ROM
   //assign flash_data = 16'hZ;
   //assign flash_address = 24'h0;
   //assign flash_ce_b = 1'b1;
   //assign flash_oe_b = 1'b1;
   //assign flash_we_b = 1'b1;
   //assign flash_reset_b = 1'b0;
   //assign flash_byte_b = 1'b1;
   // flash_sts is an input

   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
   // rs232_rxd and rs232_cts are inputs

   // PS/2 Ports
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

   // LED Displays
//   assign disp_blank = 1'b1;
//   assign disp_clock = 1'b0;
//   assign disp_rs = 1'b0;
//   assign disp_ce_b = 1'b1;
//   assign disp_reset_b = 1'b0;
//   assign disp_data_out = 1'b0;
   // disp_data_in is an input

   // Buttons, Switches, and Individual LEDs
   //lab3 assign led = 8'hFF;
   // button0, button1, button2, button3, button_enter, button_right,
```

```
   // button_left, button_down, button_up, and switches are inputs

   // User I/Os
   assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4[31:11] = 21'hZ;
        assign user4[10]=1'b1;


   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer


//                                if((busy==0)&&(counter!=0)) begin
//                                if(writing) begin
//                                        writemode<=1;
//                                        dowrite<=1;
//                                        doread<=0;
//                                        raddr<=raddr+1;
//                                        wdata<=16'hF0F0;//later put the real value
//                                        end

   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clock_65mhz), .Q(power_on_reset),
                   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // ENTER button is user reset
   wire reset,user_reset;
   debounce db1(.reset(power_on_reset),.clock(clock_65mhz),.noisy(~button_enter),.cl
ean(user_reset));
   assign reset = user_reset | power_on_reset;

   // UP and DOWN buttons for pong paddle
   wire up,down,left,but3, but4;
   debounce db2(.reset(reset),.clock(clock_65mhz),.noisy(~button_up),.clean(up));
   debounce db3(.reset(reset),.clock(clock_65mhz),.noisy(~button_down),.clean(down))
;
        debounce db4(.reset(reset),.clock(clock_65mhz),.noisy(~button_left),.clean(l
eft));
        debounce db5(.reset(reset),.clock(clock_65mhz),.noisy(~button3),.clean(but3)
);
        debounce db6(.reset(reset),.clock(clock_65mhz),.noisy(~button4),.clean(but4)
);
//
//
   // generate basic XVGA video signals
//   wire [10:0] hcount;
//   wire [9:0]  vcount;
//   wire hsync,vsync,blank;
//   xvga xvga1(.vclock(clock_65mhz),.hcount(hcount),.vcount(vcount),
```

```
//              .hsync(hsync),.vsync(vsync),.blank(blank));

   // feed XVGA signals to user's pong game
//   wire [23:0] pixel;

   // switch[1:0] selects which video generator to use:
   //  00: user's pong game
   //  01: 1 pixel outline of active video area (adjust screen controls)
   //  10: color bars
//   reg [23:0] rgb;
//   wire border = (hcount==0 | hcount==1023 | vcount==0 | vcount==767);
//   reg b,hs,vs;
//   always @(posedge clock_65mhz) begin
//      if (switch[1:0] == 2'b01) begin
//       // 1 pixel outline of visible area (white)
//       hs <= hsync;
//       vs <= vsync;
//       b <= blank;
//       rgb <= {24{border}};
//      end else if (switch[1:0] == 2'b10) begin
//       // color bars
//       hs <= hsync;
//       vs <= vsync;
//       b <= blank;
//       rgb <= {{8{hcount[8]}}, {8{hcount[7]}}, {8{hcount[6]}}} ;
//      end
//             else if (switch[1:0] == 2'b11) begin
//       // color bars
//       hs <= hsync;
//       vs <= vsync;
//       b <= blank;
//       rgb <= {wdata[15:0],8'd0};
//      end
//   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_65mhz.
   assign vga_out_red = 0;
   assign vga_out_green = 0;
   assign vga_out_blue = 0;
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_blank_b = 0;
   assign vga_out_pixel_clock = 0;
   assign vga_out_hsync = 0;
   assign vga_out_vsync = 0;



        //ALL THE FLASH STUFF OMG THIS IS THE WORST
        reg [15:0] wdata;
        reg [15:0] wdataold;
        reg writemode;
        reg dowrite;
        reg [22:0] raddr; // address of where we want to read from flash (playing fr
om flash)
        wire [15:0] frdata;     //data being read
        reg doread; // tell flash to read from memory
        wire busy; // flash is busy, don't read/write when asserted
        wire [11:0] fsmstate;
        wire dots;
        wire writing;
        wire reading;
        wire scanning;
        assign scanning= switch [5];
        assign writing = switch[7];
        assign reading = switch[6];

        //i need to creat a fifo buffer so that i can get that data together
        wire [7:0] datausbout;
        wire newout;
        reg hold;
        reg firstone=1;
```

```
        reg [7:0]firstbyte;
        reg [15:0]datatofifo;
        reg wr_en_fifo=0;
        reg rd_en_fifo=0;
        wire[15:0] dout_fifo;
        wire full;

        usb_input usb_input(clock_27mhz,reset,user4[7:0],user4[8],user4[9],datausbou
t,newout,hold,state);
        flashfifo flashfifo(clock_27mhz, datatofifo,rd_en_fifo,reset,wr_en_fifo,dout
_fifo,empty,full);

        always @(posedge clock_27mhz) begin
                if(full) hold<=1;
                else hold<=0;

                if (firstone&&newout) begin
                        firstbyte<=datausbout;
                        firstone<=0; end
                if ((!firstone)&&newout) begin
                        datatofifo<={firstbyte[7:0],datausbout[7:0]};
                        wr_en_fifo<=1;
                        firstone<=1;
                        end
                if (!((!firstone)&&newout)) wr_en_fifo<=0;
                end

        reg [4:0] flashcounter=0;
        reg [22:0] giant=0;
        always @(posedge clock_27mhz) begin
                if (up) begin
                        writemode <=1;
                        dowrite <= 0;
                        doread <= 0;
                        wdata <= 0; // initial write data = 0
                        raddr <= 0; // initial read address = 0
                        end
                else begin
                        if(busy==0) begin
                                if(writing) begin
                                        if(!empty) begin//when fifo has no info, don
t do anything. just wait
                                                writemode<=1;
                                                doread<=0;
                                                rd_en_fifo<=0;
                                                flashcounter<=1+flashcounter;//2^5 c
lock cycles between write
                                                if (flashcounter==0) begin
                                                        rd_en_fifo<=1;
                                                        dowrite<=1;
                                                        giant<=giant+1;
                                                        wdata<=dout_fifo;
                                                        end
                                                end
                                        end
                                if(reading) begin
                                        writemode<=0;
                                        doread<=1;
                                        end
                                if (scanning)begin
                                        writemode<=0;
                                        doread<=1;

                                        if (but3) raddr<=giant-switch[3:0];
                                        else raddr<=switch[3:0];
                                        end
                                end
                                else begin //busy
                                        if (writing)begin
                                                dowrite<=0;
                                                flashcounter<=1;
                                                end
                                        end
```

```
                        end
                end
        assign led = ~{writing,reading,busy,writemode,doread,dowrite,empty,switch[0]
};
        wire [63:0] data_in_display;
        assign data_in_display= {9'd0,raddr[22:0],frdata,datatofifo};
        display_16hex display_16hex (power_on_reset, clock_27mhz, data_in_display,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out);

        flash_manager flash(.clock(clock_27mhz), .reset(switch[4]), .dots(dots), .wr
itemode(writemode), .wdata(wdata),
                                        .dowrite(dowrite), .raddr(raddr), .frdata(
frdata), .doread(doread), .busy(busy),
                                        .flash_data(flash_data), .flash_address(fl
ash_address), .flash_ce_b(flash_ce_b),
                                        .flash_oe_b(flash_oe_b), .flash_we_b(flash
_we_b), .flash_reset_b(flash_reset_b),
                                        .flash_sts(flash_sts), .flash_byte_b(flash
_byte_b), .fsmstate(fsmstate));


        assign analyzer1_data = 16'd0;//frdata[15:0];
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'd0; //{busy, writemode, dowrite, doread, wdata[7:0], r
addr[3:0]};
   assign analyzer3_clock =  1'b1;//clock_27mhz;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;









//      //ALL THE FLASH STUFF OMG THIS IS THE WORST
//      reg [15:0] wdata;
//      reg [15:0] wdataold;
//      reg writemode=0;
//      reg dowrite;
//      reg [22:0] raddr; // address of where we want to read from flash (playing fr
om flash)
//      wire [15:0] frdata;     //data being read
//      reg doread; // tell flash to read from memory
//      wire busy; // flash is busy, don't read/write when asserted
//      wire [11:0] fsmstate;
//      wire dots;
//      wire writing;
//      wire reading;
//      wire scanning;
//      assign scanning= switch [5];
//      assign writing = switch[7];
//      assign reading = switch[6];
//      assign led = ~{writing,reading,busy,writemode,doread,dowrite,switch[1:0]};
//      reg [17:0] bromaddr=0;
//      wire [15:0] memdata;
//      photoholder brom(clock_27mhz,bromaddr, memdata);
//      reg isdouble=0;
//      reg [5:0]flashcounter=1;
//      always @(posedge clock_27mhz) begin
//              flashcounter<=flashcounter+1;
//              if (up) begin
//                      writemode <=1;
//                      dowrite <= 0;
//                      doread <= 0;
//                      wdata <= 0; // initial write data = 0
```

```
//                      raddr <= 0; // initial read address = 0
//                      end
//              else begin
//                      if(busy==0) begin
//                              if(writing) begin
//                                      writemode<=1;
//                                      doread<=0;
//                                      if (flashcounter==0) begin
//                                              dowrite<=0;
//                                              if (bromaddr<=600) bromaddr<=1+broma
ddr;
//                                              end
//                                      else begin
//                                              dowrite<=1;
//                                              wdata<=memdata;
//                                              end
//                                      end
//                              if(reading) begin
//                                      writemode<=0;
//                                      doread<=1;
//                                      if(frdata!=16'hFFFF) raddr<=raddr+1;
//                                      end
//                              if (scanning)begin
//                                      writemode<=0;
//                                      doread<=1;
//                                      raddr<=switch[1:0];
//                                      end
//                              end
//                      else begin //busy
//                              //if (wdata[15:0]==memdata[15:0]) dowrite<=0;
//                      end
//              end
//      end
//
//
//      wire [63:0] data_in_display;
//      assign data_in_display= {bromaddr[11:0],memdata,raddr[3:0],wdata,frdata};
//      display_16hex display_16hex (power_on_reset, clock_27mhz, data_in_display,
//              disp_blank, disp_clock, disp_rs, disp_ce_b,
//              disp_reset_b, disp_data_out);
//
//
//      flash_manager flash(.clock(clock_27mhz), .reset(up), .dots(dots), .writemode
(writemode), .wdata(wdata),
//                                      .dowrite(dowrite), .raddr(raddr), .frdata(
frdata), .doread(doread), .busy(busy),
//                                      .flash_data(flash_data), .flash_address(fl
ash_address), .flash_ce_b(flash_ce_b),
//                                      .flash_oe_b(flash_oe_b), .flash_we_b(flash
_we_b), .flash_reset_b(flash_reset_b),
//                                      .flash_sts(flash_sts), .flash_byte_b(flash
_byte_b), .fsmstate(fsmstate));
//      assign analyzer1_data = 16'd0;//frdata[15:0];
//   assign analyzer1_clock = 1'b1;
//   assign analyzer2_data = 16'h0;
//   assign analyzer2_clock = 1'b1;
//   assign analyzer3_data = {busy, writemode, dowrite, doread, wdata[7:0], raddr[3:
0]};
//   assign analyzer3_clock = clock_27mhz;
//   assign analyzer4_data = 16'h0;
//   assign analyzer4_clock = 1'b1;
endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)
//
//////////////////////////////////////////////////////////////////////////////////

//module xvga(input vclock,
//            output reg [10:0] hcount,    // pixel number on current line
//            output reg [9:0] vcount,   // line number
//            output reg vsync,hsync,blank);
```

```
//
//    // horizontal: 1344 pixels total
//    // display 1024 pixels per line
//    reg hblank,vblank;
//    wire hsyncon,hsyncoff,hreset,hblankon;
//    assign hblankon = (hcount == 1023);
//    assign hsyncon = (hcount == 1047);
//    assign hsyncoff = (hcount == 1183);
//    assign hreset = (hcount == 1343);
//
//    // vertical: 806 lines total
//    // display 768 lines
//    wire vsyncon,vsyncoff,vreset,vblankon;
//    assign vblankon = hreset & (vcount == 767);
//    assign vsyncon = hreset & (vcount == 776);
//    assign vsyncoff = hreset & (vcount == 782);
//    assign vreset = hreset & (vcount == 805);
//
//    // sync and blanking
//    wire next_hblank,next_vblank;
//    assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
//    assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
//    always @(posedge vclock) begin
//       hcount <= hreset ? 0 : hcount + 1;
//       hblank <= next_hblank;
//       hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low
//
//       vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
//       vblank <= next_vblank;
//       vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low
//
//       blank <= next_vblank | (next_hblank & ~hreset);
//    end
//endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    17:57:54 12/02/2016
// Design Name:
// Module Name:    gaussian
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements and applies a 3x3 Gaussian blur to an input matrix of
//              values. Outputs 8-bit blurred RGB. Pipelined with 3 stages.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module gaussian(
    input clk, rst,
    // red channel
    input [2:0] a0r, a1r, a2r,
    input [2:0] a7r, pix_r, a3r,
    input [2:0] a6r, a5r, a4r,
    // green channel
    input [2:0] a0g, a1g, a2g,
    input [2:0] a7g, pix_g, a3g,
    input [2:0] a6g, a5g, a4g,
    // blue channel
    input [1:0] a0b, a1b, a2b,
    input [1:0] a7b, pix_b, a3b,
    input [1:0] a6b, a5b, a4b,
    // rgb output
    output [7:0] rgb_out
  );

  // Gaussian 3x3 kernel

  //         ( 1 2 1 )
  //  1/16   ( 2 4 2 )
  //         ( 1 2 1 )

  // blur each channel separately!

  // latched input values
  reg [2:0] a0r_q, a1r_q, a2r_q, a7r_q, pix_r_q, a3r_q, a6r_q, a5r_q, a4r_q;
  reg [2:0] a0g_q, a1g_q, a2g_q, a7g_q, pix_g_q, a3g_q, a6g_q, a5g_q, a4g_q;
  reg [1:0] a0b_q, a1b_q, a2b_q, a7b_q, pix_b_q, a3b_q, a6b_q, a5b_q, a4b_q;

  // intermediate sum values
  reg [6:0] r_sum_q;
  reg [6:0] g_sum_q;
  reg [5:0] b_sum_q;

  // registered outputs
  reg [2:0] r_out_q;
  reg [2:0] g_out_q;
  reg [1:0] b_out_q;

  assign rgb_out = {r_out_q, g_out_q, b_out_q};

  always @(posedge clk) begin

    // clock 1: latch inputs

    // red channel
    {a0r_q, a1r_q, a2r_q} <= {a0r, a1r, a2r};
    {a7r_q, pix_r_q, a3r_q} <= {a7r, pix_r, a3r};
    {a6r_q, a5r_q, a4r_q} <= {a6r, a5r, a4r};
    // green channel
```

```
    {a0g_q, a1g_q, a2g_q} <= {a0g, a1g, a2g};
    {a7g_q, pix_g_q, a3g_q} <= {a7g, pix_g, a3g};
    {a6g_q, a5g_q, a4g_q} <= {a6g, a5g, a4g};
    // blue channel
    {a0b_q, a1b_q, a2b_q} <= {a0b, a1b, a2b};
    {a7b_q, pix_b_q, a3b_q} <= {a7b, pix_b, a3b};
    {a6b_q, a5b_q, a4b_q} <= {a6b, a5b, a4b};

    // clock 2: multiply by kernel values

    r_sum_q <= a0r_q+(a1r_q<<1)+a2r_q+(a7r_q<<1)+(pix_r_q<<2)+(a3r_q<<1)+a6r_q+(a5r_
q<<1)+a4r_q;
    g_sum_q <= a0g_q+(a1g_q<<1)+a2g_q+(a7g_q<<1)+(pix_g_q<<2)+(a3g_q<<1)+a6g_q+(a5g_
q<<1)+a4g_q;
    b_sum_q <= a0b_q+(a1b_q<<1)+a2b_q+(a7b_q<<1)+(pix_b_q<<2)+(a3b_q<<1)+a6b_q+(a5b_
q<<1)+a4b_q;

    // clock 3: divide by 16

    r_out_q <= r_sum_q[6:4];
    g_out_q <= g_sum_q[6:4];
    b_out_q <= b_sum_q[5:4];

  end

endmodule
```

```
`timescale 1ns / 1ps

module graphicsmaker(input pixel_clk,
                                        input [10:0] x,hcount,
                                        input [9:0] y,vcount,
                                        input [1:0] graphic,
                                        output reg[23:0] pixel);

        reg [8:0] image_addr;
        wire [8:0] image_mem;
    wire [239:0] image_bits;
    reg [7:0] mem_iter=0;
        reg [7:0] hdelay=0;
        reg hcountold;
        reg vcountold;
        reg [7:0] HEIGHT;
        reg [23:0] color;
        reg [7:0] WIDTH=240;

    always @(posedge pixel_clk) begin
                        case(graphic)
                                0:begin
                                        image_addr<=0; //Crown
                                        HEIGHT<=144; //height in memory of the COE
                                        color<=24'hF7DB24; //gold
                                 end
                                1:begin
                                        image_addr<=144;//Sunglasses
                                        HEIGHT<=44;
                                        color<=24'h070706;//jet black
                                 end
                                2:begin
                                        image_addr<=192;//Mustache
                                        HEIGHT<=38;
                                        color<=24'h331900; //dark brown
                                 end
                                3:begin
                                        image_addr<=231;//Safari Hat
                                        HEIGHT<=97;
                                        color<=24'h193300;//green brown
                                 end
                        endcase

                        hcountold<=hcount[0];//for check when hcount changes
                        vcountold<=vcount[0]; //for check when vcount changes

                        if (vcount==y) mem_iter<=0;
                        if ((~(vcount==y))&&(vcount[0]!=vcountold)&&(vcount>y)&&(vco
unt<HEIGHT+y)) mem_iter<=mem_iter+1;
                        //if we are within range and vcount has changed. then go get
 the next line in memory

                        if ((hcount[0]!=hcountold)&&(hdelay!=240)&&(hcount>=x)&&(hco
unt<=WIDTH+x)) hdelay<=hdelay+1;
                        //if we are within range and hcount has changed. then look a
t the next pixel

                        if ((hcount[0]!=hcountold)&&(hdelay==240)&&(hcount>=x)&&(hco
unt<=WIDTH+x)) hdelay<=0;
                        //need to reset so it doesnt wrap

                        if (graphic==0) begin
                                        if ((mem_iter<=16)&&(image_bits[239-hdelay[7:0]]==1)
) pixel <= 24'hCC0000; //red for jewels
                                        else if ((mem_iter>=118)&&(image_bits[239-hdelay[7:0
]]==1)) pixel <= 24'hF0D520;//Slightly sarker brim
                                        else if (image_bits[239-hdelay[7:0]]==1)pixel <= col
or; //check for zero or one. see if we should assign color
                                        else pixel <= 24'hFFFFFF;
                                        end
                        else if (graphic==1) begin
                                        if ((mem_iter>=7&&mem_iter<=28)&&((hdelay>=52&hdelay
<=96)||((hdelay>=143&&hdelay<=185)))&&(image_bits[239-hdelay[7:0]]==1)) pixel <= 24'
```

```
h404040; //make differnt black so Diana can alpha blend
                                    else if (image_bits[239-hdelay[7:0]]==1)pixel <= col
or;
                                    else pixel <= 24'hFFFFFF;
                                    end
                            else if (graphic==3) begin
                                    if ((mem_iter>=63)&&(image_bits[239-hdelay[7:0]]==1)
) pixel <= 24'h994C00; //brown string
                                    else if (image_bits[239-hdelay[7:0]]==1)pixel <= col
or;
                                    else pixel <= 24'hFFFFFF;
                                    end
                            else begin
                                    if (image_bits[239-hdelay[7:0]]==1)pixel <= color;
                                    else pixel <= 24'hFFFFFF;
                                    end
                    end
            assign image_mem=image_addr+mem_iter; //combination of the offest for specif
ic graphic and the iterator
            graphics_rom graphicholder(pixel_clk,image_mem,image_bits);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    21:21:21 12/01/2016
// Design Name:
// Module Name:    grayscale
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements conversion from RGB color to grayscale.
//              Pipelined with 3 stages.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module grayscale(
    input clk, rst,
    input [7:0] r_in, g_in, b_in,
    output [7:0] r_out, g_out, b_out
  );

  // Conversion from RGB color to grayscale
  // RGB(gray) = 0.2989R + 0.5870G + 0.1140B

  localparam R = 76;    // 0.2989
  localparam G = 150;   // 0.5870
  localparam B = 29;    // 0.1140

  reg [7:0] r0_q, g0_q, b0_q;     // latched inputs
  reg [15:0] r1_q, g1_q, b1_q;    // multiplied by R/G/B multipliers
  reg [7:0] r2_q, g2_q, b2_q;     // registered outputs

  // output assignment
  assign {r_out, g_out, b_out} = {r2_q, g2_q, b2_q};

  always @(posedge clk) begin

    // clock 1: latch inputs
    {r0_q, g0_q, b0_q} <= {r_in, g_in, b_in};

    // clock 2: multiply by R/G/B multipliers
    {r1_q, g1_q, b1_q} <= {R*r0_q, G*g0_q, B*b0_q};

    // clock 3: divide by 256 and add
    r2_q <= r1_q[15:8] + g1_q[15:8] + b1_q[15:8];
    g2_q <= r1_q[15:8] + g1_q[15:8] + b1_q[15:8];
    b2_q <= r1_q[15:8] + g1_q[15:8] + b1_q[15:8];

  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    20:33:38 11/07/2016
// Design Name:
// Module Name:    hsv2rgb
// Project Name:
// Target Devices:
// Tool versions:
// Description: Converts HSV values into RGB values. Conversion uses only
//              integer math. Pipelined with 10 stages.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module hsv2rgb(
    input clk,
    input rst,
    input [7:0] h,
    input [7:0] s,
    input [7:0] v,
    output [7:0] r,
    output [7:0] g,
    output [7:0] b
    );

    // HSV to RGB conversion using integer math
    // algorithm referenced from following source:
    // http://web.mit.edu/storborg/Public/hsvtorgb.c

    // latched inputs
    reg [7:0] h_q, s_q, v_q;

    reg [3:0] hue_region_q;         // divide hue range into 6 regions (255/6)
    reg [5:0] hue_remainder_q;      // remainder of hue_reg/43
    reg [7:0] hue_remainder255_q;   // hue_remainder*6, in [0:255] range

    reg [10:0] h6_q;                    // hue*6
    reg [7:0] hue_sum_q;                // 43*hue_region

    // temporary variables
    reg [15:0] p0_q, q0_q, t0_q;
    reg [15:0] p1_q, q1_q, t1_q;
    reg [15:0] p2_q, q2_q, t2_q;
    reg [15:0] p3_q, q3_q, t3_q;

    // delayed signals
    reg [7:0] h1_q, h2_q;                                       // delayed hue
    reg [7:0] s1_q, s2_q, s3_q, s4_q, s5_q;                     // delayed saturation
    reg [7:0] v1_q, v2_q, v3_q, v4_q, v5_q, v6_q, v7_q, v8_q; // delayed value
    reg [3:0] hrg1_q, hrg2_q, hrg3_q, hrg4_q, hrg5_q, hrg6_q; // delayed hue region

    // latched output values
    reg [7:0] r_q, g_q, b_q;

    // output assignments
    assign {r, g, b} = {r_q, g_q, b_q};

    always @(posedge clk) begin
      //if (rst) begin
      //end else begin

      // clock 1: latch inputs, reset counters
      {h_q, s_q, v_q} <= {h, s, v};

      // clock 2: multiply
      h6_q <= 6 * h_q;
```

```
        {h1_q, s1_q, v1_q} <= {h_q, s_q, v_q};

        // clock 3: calc hue region
        hue_region_q <= h6_q[10:8];
        hue_sum_q <= 6'd43 * h6_q[10:8];
        {h2_q, s2_q, v2_q} <= {h1_q, s1_q, v1_q};

        // clock 4: calc remainder
        hue_remainder_q <= h2_q - hue_sum_q;
        {s3_q, v3_q} <= {s2_q, v2_q};
        hrg1_q <= hue_region_q;

        // clock 5: multiply remainder
        hue_remainder255_q <= 6 * hue_remainder_q;
        {s4_q, v4_q} <= {s3_q, v3_q};
        hrg2_q <= hrg1_q;

        // clocks 6: calculate temporary variables
        p0_q <= 8'd255 - s4_q;
        q0_q <= hue_remainder255_q;
        t0_q <= 8'd255 - hue_remainder255_q;
        {s5_q, v5_q} <= {s4_q, v4_q};
        hrg3_q <= hrg2_q;

        // clock 7
        p1_q <= v5_q * p0_q;
        q1_q <= s5_q * q0_q;
        t1_q <= s5_q * t0_q;
        v6_q <= v5_q;
        hrg4_q <= hrg3_q;

        // clock 8
        p2_q <= p1_q;
        q2_q <= 8'd255 - q1_q[15:8];
        t2_q <= 8'd255 - t1_q[15:8];
        v7_q <= v6_q;
        hrg5_q <= hrg4_q;

        // clock 9
        p3_q <= p2_q;
        q3_q <= v7_q * q2_q;
        t3_q <= v7_q * t2_q;
        v8_q <= v7_q;
        hrg6_q <= hrg5_q;

        // clock 10: output values
        case (hrg6_q)
          0       : begin
                      r_q <= v8_q;
                      g_q <= t3_q[15:8];
                      b_q <= p3_q[15:8];
                    end
          1       : begin
                      r_q <= q3_q[15:8];
                      g_q <= v8_q;
                      b_q <= p3_q[15:8];
                    end
          2       : begin
                      r_q <= p3_q[15:8];
                      g_q <= v8_q;
                      b_q <= t3_q[15:8];
                    end
          3       : begin
                      r_q <= p3_q[15:8];
                      g_q <= q3_q[15:8];
                      b_q <= v8_q;
                    end
          4       : begin
                      r_q <= t3_q[15:8];
                      g_q <= p3_q[15:8];
                      b_q <= v8_q;
                    end
          default : begin
```

```
                    r_q <= v8_q;
                    g_q <= p3_q[15:8];
                    b_q <= q3_q[15:8];
                end
        endcase
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    20:14:16 11/20/2016
// Design Name:
// Module Name:    invert
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements a negative filter. Inverts RGB values. 1 cycle delay.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module invert(
    input clk, rst,
    input [7:0] r_in, g_in, b_in,
    output [7:0] r_out, g_out, b_out
  );

  reg [7:0] r_out_q, g_out_q, b_out_q;

  always @(posedge clk) begin
    r_out_q <= 8'd255 - r_in;
    g_out_q <= 8'd255 - g_in;
    b_out_q <= 8'd255 - b_in;
  end

  assign {r_out, g_out, b_out} = {r_out_q, g_out_q, b_out_q};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    23:46:56 12/01/2016
// Design Name:
// Module Name:    line_buf
// Project Name:
// Target Devices:
// Tool versions:
// Description: Parameterized line buffer module. Creates 2 line buffers that can
//              each store up to ELEM_LEN elements. Also creates a 3-element
//              buffer. Useful when applying a 3x3 kernel on an image. Output is
//              a set of signals that correspond to the 3x3 matrix consisting of
//              8 neighboring pixels surrounding a center pixel.
//
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module line_buf #(parameter ELEM_LEN=1) (
    input clk, rst,
    input [ELEM_LEN-1:0] pixel_in,
    output [ELEM_LEN-1:0] a0, a1, a2,
    output [ELEM_LEN-1:0] a7, pix, a3,
    output [ELEM_LEN-1:0] a6, a5, a4
  );

  `include "param.v"

  // declare line buffers
  reg [ELEM_LEN-1:0] line1 [LINE_LEN-1:0];
  reg [ELEM_LEN-1:0] line2 [LINE_LEN-1:0];
  reg [ELEM_LEN-1:0] line3 [2:0];

  // registered outputs
  reg [ELEM_LEN-1:0] a0_q, a1_q, a2_q, a7_q, pix_q, a3_q, a6_q, a5_q, a4_q;

  // output assignments
  assign {a0, a1, a2} = {a0_q, a1_q, a2_q};
  assign {a7, pix, a3} = {a7_q, pix_q, a3_q};
  assign {a6, a5, a4} = {a6_q, a5_q, a4_q} ;

  integer i1, i2, i3;

  always @(posedge clk) begin

    // shift line1
    for (i1=1; i1<LINE_LEN; i1=i1+1) line1[i1] <= line1[i1-1];
    line1[0] <= line2[LINE_LEN-1];

    // shift line2
    for (i2=1; i2<LINE_LEN; i2=i2+1) line2[i2] <= line2[i2-1];
    line2[0] <= line3[2];

    // shift line3
    for (i3=1; i3<3; i3=i3+1) line3[i3] <= line3[i3-1];
    line3[0] <= pixel_in;

    //  a0    a1    a2
    //  a7   pix    a3
    //  a6    a5    a4

    {a0_q, a1_q, a2_q} <= {line1[LINE_LEN-1], line1[LINE_LEN-2], line1[LINE_LEN-3]};
    {a7_q, pix_q, a3_q} <= {line2[LINE_LEN-1], line2[LINE_LEN-2], line2[LINE_LEN-3]}
;
    {a6_q, a5_q, a4_q} <= {line3[2], line3[1], line3[0]};

  end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk (diana96@mit.edu)
//
// Create Date:    11:51:35 11/21/2016
// Design Name:
// Module Name:    main_fsm
// Project Name:
// Target Devices:
// Tool versions:
// Description: Main FSM for the system.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module main_fsm(
    input clk, rst,
    input sw_ntsc,
    input enter,
    input store_bram,
    output [2:0] fsm_state,
    output sw_ntsc_falling
  );

  `include "param.v"

  reg [2:0] fsm_state_q = 3'b000;
  reg [2:0] next_state;

  assign fsm_state = fsm_state_q;

  //////////////////////////////////////////////////////////////////////////////////
  //
  // INPUT SIGNAL EDGE DETECTION
  //
  //////////////////////////////////////////////////////////////////////////////////

  // latched signals for edge detection
  reg sw_ntsc_d_q, enter_d_q, store_bram_d_q;

  // edge signal declarations
  wire sw_ntsc_rising; // sw_ntsc_falling; -> an output
  wire enter_rising, enter_falling;
  wire store_bram_rising, store_bram_falling;

  // edge detection for ntsc switch
  always @(posedge clk) sw_ntsc_d_q <= sw_ntsc;
  assign sw_ntsc_rising = sw_ntsc && !sw_ntsc_d_q;
  assign sw_ntsc_falling = !sw_ntsc && sw_ntsc_d_q;

  // edge detection for enter button
  always @(posedge clk) enter_d_q <= enter;
  assign enter_rising = enter && !enter_d_q;
  assign enter_falling = !enter && enter_d_q;

  // edge detection for bram switch
  always @(posedge clk) store_bram_d_q <= store_bram;
  assign store_bram_rising = store_bram && !store_bram_d_q;
  assign store_bram_falling = !store_bram && store_bram_d_q;

  //////////////////////////////////////////////////////////////////////////////////
  //
  // STATE TRANSITIONS
  //
  //////////////////////////////////////////////////////////////////////////////////

  always @(*) begin
    case (fsm_state_q)
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    16:16:22 11/21/2016
// Design Name:
// Module Name:    mover
// Project Name:
// Target Devices:
// Tool versions:
// Description: Generates x and y coordinate offsets given user button inputs.
//              Offset values are updated on the falling edge of vsync. When
//              connected to a module that generates pixel values, the outputs
//              from this module allow the user to control the movement of
//              pixel(s) across the screen. In this project, this module is
//              connected to text and graphics generator modules.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module mover(
    input clk, rst,
    input move_en,
    input up, down, left, right,
    input vsync,
    input [10:0] h_offset,
    output [10:0] x_pos,
    output [9:0] y_pos
  );

  `include "param.v"

  // The parameters below take into account timing delays from the preceding
  // modules (e.g. color space conversion modules, image enhancement). H_MIN is
  // offset to ensure that H_MIN corresponds to the top leftmost pixel on screen.

  // Display area for movement defined to be (0<x<640, 0<y<480).

  // horizontal position parameters
  //localparam H_MIN = h_offset; -> can't do this!
  localparam H_MAX = 10'd640;
  //localparam H_INIT = H_MIN;  // initial x-coordinate

  // vertical position parameters
  localparam V_MIN = 10'd0;
  localparam V_MAX = 10'd480;
  localparam V_INIT = V_MIN;  // initial y-coordinate

  // detect falling edge of vsync
  reg vsync_q;
  always @(posedge clk) vsync_q <= vsync;

  wire vsync_falling;
  assign vsync_falling = (vsync !== vsync_q) && (vsync === 1'b0);

  // registered outputs (init to approx. center)
  reg [10:0] x_pos_q = 320;
  reg [9:0] y_pos_q = 240;

  // output assignments
  assign {x_pos, y_pos} = {x_pos_q, y_pos_q};

  // determine whether current (x,y) is in defined display area
  wire x_pos_in_disp = (x_pos >= h_offset) && (x_pos < H_MAX);
  wire y_pos_in_disp = (y_pos >= V_MIN) && (y_pos < V_MAX);

  always @(posedge clk) begin
    if (rst) {x_pos_q, y_pos_q} <= {h_offset, V_INIT};
```

```
    else if (vsync_falling && move_en) begin
      // if up and down are pressed together, do nothing; otherwise, adjust vertical
 position pixel
      if (up && ~down && y_pos_in_disp) y_pos_q <= (y_pos_q == V_MIN) ? y_pos_q : y_
pos_q-1'b1;
      if (down && ~up && y_pos_in_disp) y_pos_q <= (y_pos_q == (V_MAX-1)) ? y_pos_q
: y_pos_q+1'b1;
      // if left and right are pressed together, do nothing; otherwise, adjust horiz
ontal position pixel
      if (left && ~right && x_pos_in_disp) x_pos_q <= (x_pos_q == h_offset) ? x_pos_
q : x_pos_q-1'b1;
      if (right && ~left && x_pos_in_disp) x_pos_q <= (x_pos_q == (H_MAX-1)) ? x_pos
_q : x_pos_q+1'b1;
    end
  end

endmodule
```

```
      FSM_IDLE      : next_state = (sw_ntsc_rising) ? SEL_BKGD : FSM_IDLE;

      SEL_BKGD      : next_state = (enter_rising) ? COLOR_EDITS :
                                   (sw_ntsc_falling) ? FSM_IDLE : SEL_BKGD;

      COLOR_EDITS   : next_state = (enter_rising) ? ADD_EDITS :
                                   (sw_ntsc_falling) ? FSM_IDLE : COLOR_EDITS;

      ADD_EDITS     : next_state = (store_bram_rising) ? SAVE_TO_BRAM :
                                   (sw_ntsc_falling) ? FSM_IDLE : ADD_EDITS;

      SAVE_TO_BRAM  : next_state = (enter_rising) ? SEND_TO_PC :
                                   (sw_ntsc_falling) ? FSM_IDLE : SAVE_TO_BRAM;

      SEND_TO_PC    : next_state = (sw_ntsc_falling) ? FSM_IDLE : SEND_TO_PC;

      default       : next_state = fsm_state_q;
    endcase
  end

  always @(posedge clk) begin
    if (rst) fsm_state_q <= FSM_IDLE;
    else fsm_state_q <= next_state;
  end

endmodule
```

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
//////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we);

   input          clk;   // system clock
   input          vclk;  // video clock from camera
   input [2:0]    fvh;
   input          dv;
   input [29:0]           din;
   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output         ntsc_we;       // write enable for NTSC data

   parameter      COL_START = 10'd0;
   parameter      ROW_START = 10'd0;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 * 788 XGA display

   reg [9:0]      col = 0;
   reg [9:0]      row = 0;
   reg [29:0]     vdata = 0;
   reg            vwe;
   reg            old_dv;
   reg            old_frame;      // frames are even / odd interlaced
   reg            even_odd;       // decode interlaced frame to this wire

   wire           frame = fvh[2];
   wire           frame_edge = frame & ~old_frame;

   always @ (posedge vclk) //LLC1 is reference
     begin
        old_dv <= dv;
        vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
        old_frame <= frame; //used to detect frame edge
        //on frame edge, flip even_odd
        even_odd = frame_edge ? ~even_odd : even_odd;

        if (!fvh[2])
```

```
          begin
             col <= fvh[0] ? COL_START :
                    (!fvh[2] && !fvh[1] && dv && (col < 640)) ? col + 1 : col;
             row <= fvh[1] ? ROW_START :
                    (!fvh[2] && fvh[0] && (row < 480)) ? row + 1 : row;
             vdata <= (dv && !fvh[2]) ? din : vdata;
          end
    end

   // synchronize with system clock

   reg [9:0] x[1:0],y[1:0];
   reg [29:0] data[1:0];
   reg       we[1:0];
   reg       eo[1:0];

        //delay x,y,data,we,eo by 2 clock cycles
   always @(posedge clk)
     begin
                        {x[1],x[0]} <= {x[0],col};
                        {y[1],y[0]} <= {y[0],row};
                        {data[1],data[0]} <= {data[0],vdata};
                        {we[1],we[0]} <= {we[0],vwe};
                        {eo[1],eo[0]} <= {eo[0],even_odd};
     end

   // edge detection on write enable signal
   reg old_we;
   wire we_edge = we[1] & ~old_we;
   always @(posedge clk) old_we <= we[1];

   reg [39:0] x_delay;
   reg [39:0] y_delay;
   reg [3:0] we_delay;
   reg [3:0] eo_delay;

   always @ (posedge clk)
   begin
     x_delay <= {x_delay[29:0], x[1]};
     y_delay <= {y_delay[29:0], y[1]};
     we_delay <= {we_delay[2:0], we[1]};
     eo_delay <= {eo_delay[2:0], eo[1]};
   end

   // compute address to store data in
   wire [8:0] y_addr = y_delay[38:30];
        wire [9:0] x_addr = x_delay[39:30];
   wire [18:0] myaddr = {y_addr[7:0], eo_delay[3], x_addr[9:0]};

        reg [18:0] ntsc_addr;
        reg [35:0] ntsc_data;
        wire ntsc_we;

        assign ntsc_we = we_delay[3];
   always @(posedge clk)
     if ( ntsc_we )
       begin
                        ntsc_addr <= myaddr;
                        ntsc_data <= {5'b0, data[1]};
       end

endmodule // ntsc_to_zbt
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    18:13:32 11/29/2016
// Design Name:
// Module Name:    param
// Project Name:
// Target Devices:
// Tool versions:
// Description: Defines values for various system parameters.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Display Parameters
parameter H_MAX_NTSC = 11'd640;       // frame size of NTSC camera input
parameter V_MAX_NTSC = 10'd480;       // frame size of NTSC camera input
parameter VGA_HCOUNT_MAX = 11'd1056;      // for 800x600 resolution
parameter VGA_VCOUNT_MAX = 10'd628;       // for 800x600 resolution
parameter XVGA_HCOUNT_MAX = 11'd1344;      // for 1024x768 resolution
parameter XVGA_VCOUNT_MAX = 10'd806;       // for 1024x768 resolution

// FSM States
parameter FSM_IDLE = 3'b000;
parameter SEL_BKGD = 3'b001;
parameter COLOR_EDITS = 3'b010;
parameter ADD_EDITS = 3'b011;
parameter SAVE_TO_BRAM = 3'b100;
parameter SEND_TO_PC = 3'b101;

// BRAM States
parameter BRAM_IDLE = 2'b00;
parameter CAPTURE_FRAME = 2'b01;
parameter WRITING_FRAME = 2'b10;
parameter READING_FRAME = 2'b11;

// Backgrounds
parameter PARIS = 3'b000;
parameter ROME = 3'b001;
parameter AMAZON = 3'b010;
parameter LONDON = 3'b011;
parameter NO_BKD = 3'b100;

// Graphics
parameter MUSTACHE = 2'b00;
parameter SUNGLASSES = 2'b01;
parameter SAFARI_HAT = 2'b10;
parameter CROWN = 2'b11;

// Constants
parameter CUSTOM_TEXT_MAXLEN = 20;

// Filter Types
parameter SEPIA = 3'b000;
parameter INVERT = 3'b001;
parameter EDGE = 3'b010;
parameter CARTOON = 3'b011;
parameter GRAYSCALE = 3'b100;

// Pipeline Stages
parameter YCRCB2RGB_DLY = 4;
parameter RGB2HSV_DLY = 23;
parameter THRESHOLD_DLY = 1;
parameter HSV2RGB_DLY = 10;
parameter ENHANCE_DLY = 1;
```

```
parameter INVERT_DLY = 1;
parameter SEPIA_DLY = 4;
parameter GRAYSCALE_DLY = 3;

parameter LINE_LEN = VGA_HCOUNT_MAX;
parameter LINE_BUF_DLY = LINE_LEN*2 + 3 + 1;
parameter SOBEL_OP_DLY = 4;
parameter EDGE_DET_DLY = 1;
parameter GAUSSIAN_DLY = 3;

parameter SOBEL_DLY = GRAYSCALE_DLY + LINE_BUF_DLY + SOBEL_OP_DLY + EDGE_DET_DLY;

// Edge Detection
parameter GRADIENT_EDGE_THRESHOLD = 15'd50; // for sketch filter effect
parameter CARTOON_EDGE_THRESHOLD = 15'd100; // for cartoon filter effect

// Sync Delay Values
parameter SYNC_DLY = YCRCB2RGB_DLY + RGB2HSV_DLY + THRESHOLD_DLY + HSV2RGB_DLY + ENH
ANCE_DLY + 1;

parameter SYNC_DLY_SEP = SYNC_DLY + SEPIA_DLY;
parameter SYNC_DLY_INV = SYNC_DLY + INVERT_DLY;
parameter SYNC_DLY_GRY = SYNC_DLY + GRAYSCALE_DLY;
parameter SYNC_DLY_SBL = SYNC_DLY + SOBEL_DLY; // applies to both sketch and cartoon
 filter effects

parameter MAX_SYNC_DLY = SYNC_DLY_SBL; // max # of stages in pipelined design

// BRAM Storage
//parameter H_OFFSET = SYNC_DLY;       // -> redefined as conditional assignment in
zbt_6111_sample
parameter V_OFFSET = 10'd0;
parameter H_MAX_DISPLAY = 11'd640;   // for storage in BRAM
parameter V_MAX_DISPLAY = 10'd400;   // for storage in BRAM
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    22:29:17 11/17/2016
// Design Name:
// Module Name:    pixel_sel
// Project Name:
// Target Devices:
// Tool versions:
// Description: Pixel selector module. Instantiates and interconnects the
//              color space conversion modules, the image enhacenment and
//              filter effects modules, as well as the text and graphics
//              generation modules. Processes user button and switch inputs
//              and selects output VGA pixels accordingly.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module pixel_sel #(parameter TEXT_LEN_MAX=20) (
    input clk, reset,
    // states
    input [2:0] fsm_state,
    input [1:0] bram_state,
    // user switches
    input sw_ntsc,
    input store_bram,
    input enhance_en,
    input filters_en,
    input text_en,
    input graphics_en,
    input move_text_en,
    input move_graphics_en,
    input custom_text_en,
    // user buttons
    input up, down, left, right,
    input select0, select1, select2, select3,
    input [2:0] background,
    // custom text gen
    input [5:0] num_char,
    input char_array_rdy,
    input [TEXT_LEN_MAX*8-1:0] char_array,
    // pixel values
    input [29:0] vr_pixel,
    input [7:0] bram_dout,
    input [23:0] vr_bkgd_color,
    // VGA timing
    input [10:0] hcount,
    input [9:0] vcount,
    input blank,
    input hsync,
    input vsync,
    input [10:0] h_offset,
    input in_display_bram,
    // VGA outputs
    output [23:0] pixel_out,
    output blank_out,
    output hsync_out,
    output vsync_out,
    output [10:0] text_x_pos,
    output [9:0] text_y_pos,
    output [10:0] graphics_x_pos,
    output [9:0] graphics_y_pos,
    // hex display
    output [7:0] thr_range, h_thr, s_thr, v_thr,
    // image enhacement
    output [7:0] s_offset, v_offset,
    output s_dir, v_dir,
```

```
    // user selections
    output[2:0] selected_filter,
    output [1:0] selected_graphic
  );

  `include "param.v"

  //wire [2:0] selected_filter;  // used in determining filter module latency -> out
put
  wire [23:0] vga_rgb_out;     // connected to VGA pixel output

  // ***********************************************
  // YCrCb to RGB Conversion
  // ***********************************************

  wire [23:0] vr_pixel_color;

  ycrcb2rgb ycrcb2rgb_conv(
    .Y    (vr_pixel[29:20]),
    .Cr   (vr_pixel[19:10]),
    .Cb   (vr_pixel[9:0]),
    .R    (vr_pixel_color[23:16]),
    .G    (vr_pixel_color[15:8]),
    .B    (vr_pixel_color[7:0]),
    .clk  (clk),
    .rst  (1'b0)
   );

  // ***********************************************
  // RGB to HSV Conversion
  // ***********************************************

  wire [23:0] pixel_hsv_out;
  wire [23:0] bkgd_hsv_out;

  rgb2hsv rgb2hsv_conv1(
    .clock(clk),
    .reset(reset),
    .r(vr_pixel_color[23:16]),
    .g(vr_pixel_color[15:8]),
    .b(vr_pixel_color[7:0]),
    .h(pixel_hsv_out[23:16]),
    .s(pixel_hsv_out[15:8]),
    .v(pixel_hsv_out[7:0])
  );

  rgb2hsv rgb2hsv_conv2(
    .clock(clk),
    .reset(reset),
    .r(vr_bkgd_color[23:16]),
    .g(vr_bkgd_color[15:8]),
    .b(vr_bkgd_color[7:0]),
    .h(bkgd_hsv_out[23:16]),
    .s(bkgd_hsv_out[15:8]),
    .v(bkgd_hsv_out[7:0])
  );

  // ***********************************************
  // HSV to RGB Conversion
  // ***********************************************

  wire [23:0] pixel_hsv_in;
  wire [23:0] pixel_rgb_out;

  hsv2rgb hsv2rgb_conv(
    .clk(clk),
    .rst(reset),
    .h(pixel_hsv_in[23:16]),
    .s(pixel_hsv_in[15:8]),
    .v(pixel_hsv_in[7:0]),
    .r(pixel_rgb_out[23:16]),
    .g(pixel_rgb_out[15:8]),
    .b(pixel_rgb_out[7:0])
```

```
  );

  // ************************************************
  // Chroma-Key Compositing
  // ************************************************

  wire chroma_key_match; // HSV values match threshholds
  wire [23:0] hsv_chr_out;

  // signal allowing users to adjust HSV threshholds
  wire adjust_thr_en = (fsm_state == SEL_BKGD);

  chroma_key chroma_key1(
    .clk              (clk),
    .rst              (reset),
    .vsync            (vsync),
    .hsv_chr_in       (pixel_hsv_out),
    .up               (up),
    .down             (down),
    .left             (left),
    .right            (right),
    .adjust_thr_en    (adjust_thr_en),
    .range            (thr_range),
    .h_nom            (h_thr),
    .s_nom            (s_thr),
    .v_nom            (v_thr),
    .hsv_chr_out      (hsv_chr_out),
    .chroma_key_match (chroma_key_match)
  );

  wire [23:0] bkgd_pixel_out = (background == NO_BKD) ? 24'h000000 : bkgd_hsv_out;

  wire [23:0] chr_pixel_out;     // chroma-keyed pixel
  //assign chr_pixel_out = (chroma_key_match) ? 24'hD5FFFF : hsv_chr_out;
  assign chr_pixel_out = (chroma_key_match) ? bkgd_pixel_out : hsv_chr_out;

  // ************************************************
  // Image Enhancement
  // ************************************************

  // signal allowing users to adjust saturation and brightness values
  wire enhance_user_in_en = enhance_en && (fsm_state == COLOR_EDITS);

  enhance enhance1(
    .clk                (clk),
    .rst                (reset),
          .vsync              (vsync),
    .enhance_en         (enhance_en),
    .enhance_user_in_en (enhance_user_in_en),
    .inc_saturation     (up),
    .dec_saturation     (down),
    .inc_brightness     (right),
    .dec_brightness     (left),
    .hsv_in             (chr_pixel_out),
    .hsv_out            (pixel_hsv_in),
    .s_offset           (s_offset),
    .v_offset           (v_offset),
    .s_dir              (s_dir),
    .v_dir              (v_dir)
  );

  // ************************************************
  // Filter Effects
  // ************************************************

  wire [23:0] pixel_filtered; // pixel output after filter effects

  // signal allowing users to select filter effect
  wire filters_user_in_en = filters_en && (fsm_state == COLOR_EDITS);

  filters filters1(
    .clk                (clk),
    .rst                (reset),
```

```
    .filters_en           (filters_en),
    .filters_user_in_en   (filters_user_in_en),
    .select0              (select0),
    .select1              (select1),
    .select2              (select2),
    .select3              (select3),
    .hcount               (hcount),
    .vcount               (vcount),
    .rgb_in               (pixel_rgb_out),
    .rgb_out              (pixel_filtered),
    .filter               (selected_filter)
  );

  // ************************************************
  // Text Overlay
  // ************************************************

  // Text Movement
  wire text_move_enable = text_en && move_text_en && (fsm_state == ADD_EDITS);

  mover text_mover(
    .clk      (clk),
    .rst      (reset),
    .move_en  (text_move_enable),
    .up       (up),
    .down     (down),
    .left     (left),
    .right    (right),
    .vsync    (vsync),
    .h_offset (h_offset),
    .x_pos    (text_x_pos),
    .y_pos    (text_y_pos)
  );

  // Text Crosshair
  reg [23:0] text_crosshair_pixel_q;
  always @(posedge clk) begin
    if (text_en && ((hcount == text_x_pos) || (vcount == text_y_pos)))
      text_crosshair_pixel_q <= 24'hFF0000;
    else text_crosshair_pixel_q <= 24'h000000;
  end

  // Text Generation
  wire [23:0] text_gen_pixel;

  stringmaker #(CUSTOM_TEXT_MAXLEN) text_gen(
    .clk        (clk),
    .x          (text_x_pos),
    .hcount     (hcount),
    .y          (text_y_pos),
    .vcount     (vcount),
    .background (background[1:0]),
    .numchar    (num_char),
    .ready      (char_array_rdy),
    .string     (char_array),
    .custom     (custom_text_en),
    .pixel      (text_gen_pixel)
  );

  // ************************************************
  // Graphics Overlay
  // ************************************************

  // Graphics Movement
  wire graphics_move_enable = graphics_en && move_graphics_en && (fsm_state == ADD_E
DITS);

  mover graphics_mover(
    .clk      (clk),
    .rst      (reset),
    .move_en  (graphics_move_enable),
    .up       (up),
    .down     (down),
```

```
    .left     (left),
    .right    (right),
    .vsync    (vsync),
    .h_offset (h_offset),
    .x_pos    (graphics_x_pos),
    .y_pos    (graphics_y_pos)
  );

  // Graphics Crosshair
  reg [23:0] graphics_crosshair_pixel_q;
  always @(posedge clk) begin
    if (graphics_en && ((hcount == graphics_x_pos) || (vcount == graphics_y_pos)))
      graphics_crosshair_pixel_q <= 24'h0000FF;
    else graphics_crosshair_pixel_q <= 24'h000000;
  end

  // Graphics Generation
  wire [23:0] graphics_gen_pixel;
  wire [1:0] graphics_sel;

  reg [1:0] graphics_sel_q = 0;
  assign selected_graphic = graphics_sel_q;

  always @(posedge clk) begin
    if ((fsm_state == ADD_EDITS) && select0) graphics_sel_q <= MUSTACHE;
    if ((fsm_state == ADD_EDITS) && select1) graphics_sel_q <= SUNGLASSES;
    if ((fsm_state == ADD_EDITS) && select2) graphics_sel_q <= SAFARI_HAT;
    if ((fsm_state == ADD_EDITS) && select3) graphics_sel_q <= CROWN;
  end

  graphicsmaker graphics_gen(
    .pixel_clk  (clk),
    .x          (graphics_x_pos),
    .hcount     (hcount),
    .y          (graphics_y_pos),
    .vcount     (vcount),
    .graphic    (selected_graphic),
    .pixel      (graphics_gen_pixel)
  );

  // ************************************************
  // Delay Sync Signals (Shift Registers)
  // ************************************************

  // VGA sync timing signals are delayed in order to take into account
  // the pipelined nature of all of the image processing modules; the
  // length of the delay required depends on which filter is selected.
  // Delays are defined in the parameters file - param.v

  // shift registers that can accomodate longest possible delay
  reg [0:0] hsync_shift_reg[MAX_SYNC_DLY-1:0];
  reg [0:0] vsync_shift_reg[MAX_SYNC_DLY-1:0];
  reg [0:0] blank_shift_reg[MAX_SYNC_DLY-1:0];

  integer i;

  always @(posedge clk) begin
    hsync_shift_reg[0] <= hsync;
    vsync_shift_reg[0] <= vsync;
    blank_shift_reg[0] <= blank;

    for (i=1; i<MAX_SYNC_DLY; i=i+1) begin
      hsync_shift_reg[i] <= hsync_shift_reg[i-1];
      vsync_shift_reg[i] <= vsync_shift_reg[i-1];
      blank_shift_reg[i] <= blank_shift_reg[i-1];
    end
  end

  // ************************************************
  // Output VGA Pixel
  // ************************************************

  // overlapping pixels: text > graphics > filtered/processed pixel
```

```
  // text and graphics are generated separately on a white background, so the
  // non-white pixels correspond to the text and graphics being overlayed

  assign vga_rgb_out = (text_en && (text_gen_pixel != 24'hFFFFFF)) ? text_gen_pixel
:
                       (graphics_en && (graphics_gen_pixel != 24'hFFFFFF)) ? graphi
cs_gen_pixel : pixel_filtered;

  reg [23:0] pixel_out_q; // registered output

  wire [23:0] bram_dout_24bit = {bram_dout[7:5],5'd0,bram_dout[4:2],5'd0,bram_dout[1
:0],6'd0};

  always @(posedge clk) begin
    // if reading from BRAM and displaying stored data, display 24-bit BRAM output
    // otherwise, if reading from BRAM and transmitting data to PC, display black sc
reen
    // otherwise, display VGA RGB pixel or the start splash screen
    if ((bram_state == READING_FRAME) && !(fsm_state == SEND_TO_PC))
      pixel_out_q <= in_display_bram ? bram_dout_24bit : 24'hFFFFFF;
    else pixel_out_q <= sw_ntsc ? 24'h000000 : (fsm_state == SEND_TO_PC) ? 24'h00000
0 : vga_rgb_out;
  end

  // Output Signal Assignments
  assign pixel_out = pixel_out_q;

  // ************************************************
  // Delay Sync Signals (Assignments)
  // ************************************************

  // VGA sync signals are delayed according to which filter is selected.

  assign blank_out = (!filters_en) ? blank_shift_reg[SYNC_DLY-1] :
                       (selected_filter == SEPIA) ? blank_shift_reg[SYNC_DLY_SEP-1]
 :
                       (selected_filter == INVERT) ? blank_shift_reg[SYNC_DLY_INV-1
] :
                       (selected_filter == GRAYSCALE) ? blank_shift_reg[SYNC_DLY_GR
Y-1] :
                       (selected_filter == EDGE) ? blank_shift_reg[SYNC_DLY_SBL-1]
:
                       (selected_filter == CARTOON) ? blank_shift_reg[SYNC_DLY_SBL-
1] :
                        blank_shift_reg[SYNC_DLY-1];

  assign hsync_out = (!filters_en) ? hsync_shift_reg[SYNC_DLY-1] :
                       (selected_filter == SEPIA) ? hsync_shift_reg[SYNC_DLY_SEP-1]
 :
                       (selected_filter == INVERT) ? hsync_shift_reg[SYNC_DLY_INV-1
] :
                       (selected_filter == GRAYSCALE) ? hsync_shift_reg[SYNC_DLY_GR
Y-1] :
                       (selected_filter == EDGE) ? hsync_shift_reg[SYNC_DLY_SBL-1]
:
                       (selected_filter == CARTOON) ? hsync_shift_reg[SYNC_DLY_SBL-
1] :
                        hsync_shift_reg[SYNC_DLY-1];

  assign vsync_out = (!filters_en) ? vsync_shift_reg[SYNC_DLY-1] :
                       (selected_filter == SEPIA) ? vsync_shift_reg[SYNC_DLY_SEP-1]
 :
                       (selected_filter == INVERT) ? vsync_shift_reg[SYNC_DLY_INV-1
] :
                       (selected_filter == GRAYSCALE) ? vsync_shift_reg[SYNC_DLY_GR
Y-1] :
                       (selected_filter == EDGE) ? vsync_shift_reg[SYNC_DLY_SBL-1]
:
                       (selected_filter == CARTOON) ? vsync_shift_reg[SYNC_DLY_SBL-
1] :
                        vsync_shift_reg[SYNC_DLY-1];
endmodule
```

```
//
// File:   ps2_kbd.v
// Date:   24-Oct-05
// Author: C. Terman / I. Chuang
//
// PS2 keyboard input for 6.111 labkit
//
// INPUTS:
//
//   clock_27mhz   - master clock
//   reset         - active high
//   clock         - ps2 interface clock
//   data          - ps2 interface data
//
// OUTPUTS:
//
//   ascii         - 8 bit ascii code for current character
//   ascii_ready   - one clock cycle pulse indicating new char received


//////////////////////////////////////////////////////////////////////////////

module ps2_ascii_input(clock_27mhz, reset, clock, data, ascii, ascii_ready);

   // module to generate ascii code for keyboard input
   // this is module works synchronously with the system clock

   input clock_27mhz;
   input reset;          // Active high asynchronous reset
   input clock;          // PS/2 clock
   input data;           // PS/2 data
   output [7:0] ascii;   // ascii code (1 character)
   output ascii_ready;   // ascii ready (one clock_27mhz cycle active high)

   reg [7:0]   ascii_val;       // internal combinatorial ascii decoded value
   reg [7:0]   lastkey;         // last keycode
   reg [7:0]   curkey;          // current keycode
   reg [7:0]   ascii;           // ascii output (latched & synchronous)
   reg         ascii_ready;     // synchronous one-cycle ready flag

   // get keycodes

   wire        fifo_rd;         // keyboard read request
   wire [7:0]  fifo_data;       // keyboard data
   wire        fifo_empty;      // flag: no keyboard data
   wire        fifo_overflow;   // keyboard data overflow

   ps2 myps2(reset, clock_27mhz, clock, data, fifo_rd, fifo_data,
             fifo_empty,fifo_overflow);

   assign      fifo_rd = ~fifo_empty;   // continous read
   reg         key_ready;

   always @(posedge clock_27mhz)
     begin

        // get key if ready

        curkey <= ~fifo_empty ? fifo_data : curkey;
        lastkey <= ~fifo_empty ? curkey : lastkey;
        key_ready  <= ~fifo_empty;

        // raise ascii_ready for last key which was read

        ascii_ready <= key_ready & ~(curkey[7]|lastkey[7]);
        ascii <=  (key_ready & ~(curkey[7]|lastkey[7])) ? ascii_val : ascii;

     end

   always @(curkey) begin //convert PS/2 keyboard make code ==> ascii code
     case (curkey)
       8'h1C: ascii_val = 8'h41;                //A
       8'h32: ascii_val = 8'h42;                //B
```

```
        8'h21: ascii_val = 8'h43;               //C
        8'h23: ascii_val = 8'h44;               //D
        8'h24: ascii_val = 8'h45;               //E
        8'h2B: ascii_val = 8'h46;               //F
        8'h34: ascii_val = 8'h47;               //G
        8'h33: ascii_val = 8'h48;               //H
        8'h43: ascii_val = 8'h49;               //I
        8'h3B: ascii_val = 8'h4A;               //J
        8'h42: ascii_val = 8'h4B;               //K
        8'h4B: ascii_val = 8'h4C;               //L
        8'h3A: ascii_val = 8'h4D;               //M
        8'h31: ascii_val = 8'h4E;               //N
        8'h44: ascii_val = 8'h4F;               //O
        8'h4D: ascii_val = 8'h50;               //P
        8'h15: ascii_val = 8'h51;               //Q
        8'h2D: ascii_val = 8'h52;               //R
        8'h1B: ascii_val = 8'h53;               //S
        8'h2C: ascii_val = 8'h54;               //T
        8'h3C: ascii_val = 8'h55;               //U
        8'h2A: ascii_val = 8'h56;               //V
        8'h1D: ascii_val = 8'h57;               //W
        8'h22: ascii_val = 8'h58;               //X
        8'h35: ascii_val = 8'h59;               //Y
        8'h1A: ascii_val = 8'h5A;               //Z

        8'h45: ascii_val = 8'h30;               //0
        8'h16: ascii_val = 8'h31;               //1
        8'h1E: ascii_val = 8'h32;               //2
        8'h26: ascii_val = 8'h33;               //3
        8'h25: ascii_val = 8'h34;               //4
        8'h2E: ascii_val = 8'h35;               //5
        8'h36: ascii_val = 8'h36;               //6
        8'h3D: ascii_val = 8'h37;               //7
        8'h3E: ascii_val = 8'h38;               //8
        8'h46: ascii_val = 8'h39;               //9

        8'h0E: ascii_val = 8'h60;               // `
        8'h4E: ascii_val = 8'h2D;               // -
        8'h55: ascii_val = 8'h3D;               // =
        8'h5C: ascii_val = 8'h5C;               // \
        8'h29: ascii_val = 8'h20;               // (space)
        8'h54: ascii_val = 8'h5B;               // [
        8'h5B: ascii_val = 8'h5D;               // ]
        8'h4C: ascii_val = 8'h3B;               // ;
        8'h52: ascii_val = 8'h27;               // '
        8'h41: ascii_val = 8'h2C;               // ,
        8'h49: ascii_val = 8'h2E;               // .
        8'h4A: ascii_val = 8'h2F;               // /

        8'h5A: ascii_val = 8'h0D;               // enter (CR)
        8'h66: ascii_val = 8'h08;               // backspace

        //  8'hF0: ascii_val = 8'hF0;           // BREAK CODE

        default: ascii_val = 8'h23;             // #
      endcase
    end
endmodule // ps2toascii

////////////////////////////////////////////////////////////////////////////////
// new synchronous ps2 keyboard driver, with built-in fifo, from Chris Terman

module ps2(reset, clock_27mhz, ps2c, ps2d, fifo_rd, fifo_data,
           fifo_empty,fifo_overflow);

   input clock_27mhz,reset;
   input ps2c;                  // ps2 clock
   input ps2d;                  // ps2 data
   input fifo_rd;               // fifo read request (active high)
   output [7:0] fifo_data;      // fifo data output
   output       fifo_empty;     // fifo empty (active high)
   output       fifo_overflow;  // fifo overflow - too much kbd input
```

```
  reg [3:0] count;      // count incoming data bits
  reg [9:0] shift;      // accumulate incoming data bits

  reg [7:0] fifo[7:0];   // 8 element data fifo
  reg fifo_overflow;
  reg [2:0] wptr,rptr;   // fifo write and read pointers

  wire [2:0] wptr_inc = wptr + 1;

  assign fifo_empty = (wptr == rptr);
  assign fifo_data = fifo[rptr];

  // synchronize PS2 clock to local clock and look for falling edge
  reg [2:0] ps2c_sync;
  always @ (posedge clock_27mhz) ps2c_sync <= {ps2c_sync[1:0],ps2c};
  wire sample = ps2c_sync[2] & ~ps2c_sync[1];

  always @ (posedge clock_27mhz) begin
    if (reset) begin
      count <= 0;
      wptr <= 0;
      rptr <= 0;
      fifo_overflow <= 0;
    end
    else if (sample) begin
           // order of arrival: 0,8 bits of data (LSB first),odd parity,1
           if (count==10) begin
              // just received what should be the stop bit
              if (shift[0]==0 && ps2d==1 && (^shift[9:1])==1) begin
                 fifo[wptr] <= shift[8:1];
                 wptr <= wptr_inc;
                 fifo_overflow <= fifo_overflow | (wptr_inc == rptr);
              end
              count <= 0;
           end else begin
              shift <= {ps2d,shift[9:1]};
              count <= count + 1;
           end
         end
    // bump read pointer if we're done with current value.
    // Read also resets the overflow indicator
    if (fifo_rd && !fifo_empty) begin
      rptr <= rptr + 1;
      fifo_overflow <= 0;
    end
  end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// ramclock module

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
//////////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
                clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                 // Reference clock input
   output fpga_clock;               // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
   input  clock_feedback_in;        // Output to feedback trace
   output clock_feedback_out;       // Input from feedback trace
   output locked;                   // Indicates that clock outputs are stable

   wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

   //////////////////////////////////////////////////////////////////////////////

   //To force ISE to compile the ramclock, this line has to be removed.
   //IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

        assign ref_clk = ref_clock;

   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

   DCM int_dcm (.CLKFB(fpga_clock),
                .CLKIN(ref_clk),
                .RST(dcm_reset),
                .CLK0(fpga_clk),
                .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

   BUFG ext_buf (.O(ram_clock), .I(ram_clk));

   IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

   DCM ext_dcm (.CLKFB(fb_clk),
                .CLKIN(ref_clk),
```

```
                    .RST(dcm_reset),
                    .CLK0(ram_clk),
                    .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

   SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
                     .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   // synthesis attribute init of dcm_rst_sr is "000F";


   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Kevin Zheng Class of 2012
//           Dept of Electrical Engineering &  Computer Science
//
// Create Date:    18:45:01 11/10/2010
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module rgb2hsv(clock, reset, r, g, b, h, s, v);
                input wire clock;
                input wire reset;
                input wire [7:0] r;
                input wire [7:0] g;
                input wire [7:0] b;
                output reg [7:0] h;
                output reg [7:0] s;
                output reg [7:0] v;
                reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
                reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
                reg [7:0] my_r, my_g, my_b;
                reg [7:0] min, max, delta;
                reg [15:0] s_top;
                reg [15:0] s_bottom;
                reg [15:0] h_top;
                reg [15:0] h_bottom;
                wire [15:0] s_quotient;
                wire [15:0] s_remainder;
                wire s_rfd;
                wire [15:0] h_quotient;
                wire [15:0] h_remainder;
                wire h_rfd;
                reg [7:0] v_delay [19:0];
                reg [18:0] h_negative;
                reg [15:0] h_add [18:0];
                reg [4:0] i;
                // Clocks 4-18: perform all the divisions
                //the s_divider (16/16) has delay 18
                //the hue_div (16/16) has delay 18

    divider hue_div1(
                .clk(clock),
                .dividend(s_top),
                .divisor(s_bottom),
                .quotient(s_quotient),
                // note: the "fractional" output was originally named "remainder" in
 this
                // file -- it seems coregen will name this output "fractional" even
if
                // you didn't select the remainder type as fractional.
                .fractional(s_remainder),
                .rfd(s_rfd)
                );
                divider hue_div2(
                .clk(clock),
                .dividend(h_top),
                .divisor(h_bottom),
                .quotient(h_quotient),
                .fractional(h_remainder),
                .rfd(h_rfd)
                );
```

```
                always @ (posedge clock) begin

                        // Clock 1: latch the inputs (always positive)
                        {my_r, my_g, my_b} <= {r, g, b};

                        // Clock 2: compute min, max
                        {my_r_delay1, my_g_delay1, my_b_delay1} <= {my_r, my_g, my_b
};

                        if((my_r >= my_g) && (my_r >= my_b)) //(B,S,S)
                                max <= my_r;
                        else if((my_g >= my_r) && (my_g >= my_b)) //(S,B,S)
                                max <= my_g;
                        else    max <= my_b;

                        if((my_r <= my_g) && (my_r <= my_b)) //(S,B,B)
                                min <= my_r;
                        else if((my_g <= my_r) && (my_g <= my_b)) //(B,S,B)
                                min <= my_g;
                        else
                                min <= my_b;

                        // Clock 3: compute the delta
                        {my_r_delay2, my_g_delay2, my_b_delay2} <= {my_r_delay1, my_
g_delay1, my_b_delay1};
                        v_delay[0] <= max;
                        delta <= max - min;

                        // Clock 4: compute the top and bottom of whatever divisions
 we need to do
                        s_top <= 8'd255 * delta;
                        s_bottom <= (v_delay[0]>0)?{8'd0, v_delay[0]}: 16'd1;


                        if(my_r_delay2 == v_delay[0]) begin
                                h_top <= (my_g_delay2 >= my_b_delay2)?(my_g_delay2 -
 my_b_delay2) * 8'd255:(my_b_delay2 - my_g_delay2) * 8'd255;
                                h_negative[0] <= (my_g_delay2 >= my_b_delay2)?0:1;
                                h_add[0] <= 16'd0;
                        end
                        else if(my_g_delay2 == v_delay[0]) begin
                                h_top <= (my_b_delay2 >= my_r_delay2)?(my_b_delay2 -
 my_r_delay2) * 8'd255:(my_r_delay2 - my_b_delay2) * 8'd255;
                                h_negative[0] <= (my_b_delay2 >= my_r_delay2)?0:1;
                                h_add[0] <= 16'd85;
                        end
                        else if(my_b_delay2 == v_delay[0]) begin
                                h_top <= (my_r_delay2 >= my_g_delay2)?(my_r_delay2 -
 my_g_delay2) * 8'd255:(my_g_delay2 - my_r_delay2) * 8'd255;
                                h_negative[0] <= (my_r_delay2 >= my_g_delay2)?0:1;
                                h_add[0] <= 16'd170;
                        end

                        h_bottom <= (delta > 0)?delta * 8'd6:16'd6;


                        //delay the v and h_negative signals 18 times
                        for(i=1; i<19; i=i+1) begin
                                v_delay[i] <= v_delay[i-1];
                                h_negative[i] <= h_negative[i-1];
                                h_add[i] <= h_add[i-1];
                        end

                        v_delay[19] <= v_delay[18];
                        //Clock 22: compute the final value of h
                        //depending on the value of h_delay[18], we need to subtract
 255 from it to make it come back around the circle
                        if(h_negative[18] && (h_quotient > h_add[18])) begin
                                h <= 8'd255 - h_quotient[7:0] + h_add[18];
                        end
                        else if(h_negative[18]) begin
                                h <= h_add[18] - h_quotient[7:0];
                        end
```

```
                        else begin
                                h <= h_quotient[7:0] + h_add[18];
                        end

                        //pass out s and v straight
                        s <= s_quotient;
                        v <= v_delay[19];
                end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    18:51:21 11/20/2016
// Design Name:
// Module Name:    sepia
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements RGB to sepia tone conversion. Pipelined with 4 stages.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sepia(
    input clk, rst,
    input [7:0] r_in, g_in, b_in,
    output [7:0] r_out, g_out, b_out
  );

  // Sepia conversion formulas obtained from
  // https://www.dyclassroom.com/image-processing-project/
  // how-to-convert-a-color-image-into-sepia-image

  localparam TR_R = 8'd101; // 0.393
  localparam TR_G = 8'd197; // 0.769
  localparam TR_B = 8'd48;  // 0.189

  localparam TG_R = 8'd89;  // 0.349
  localparam TG_G = 8'd176; // 0.686
  localparam TG_B = 8'd43;  // 0.168

  localparam TB_R = 8'd70;  // 0.272
  localparam TB_G = 8'd137; // 0.534
  localparam TB_B = 8'd33;  // 0.131

  reg [7:0] r0_q, g0_q, b0_q;
  reg [15:0] trr_q, trg_q, trb_q;   // to calculate tr
  reg [15:0] tgr_q, tgg_q, tgb_q;   // to calculate tg
  reg [15:0] tbr_q, tbg_q, tbb_q;   // to calculate tb
  reg [8:0] tr_q, tg_q, tb_q;
  reg [7:0] rf_q, gf_q, bf_q;       // final values

  // output assignments
  assign {r_out, g_out, b_out} = {rf_q, gf_q, bf_q};

  always @(posedge clk) begin

    // clock 1: latch inputs
    {r0_q, g0_q, b0_q} <= {r_in, g_in, b_in};

    // clock 2: multiplications
    trr_q <= r0_q * TR_R;
    trg_q <= g0_q * TR_G;
    trb_q <= b0_q * TR_B;

    tgr_q <= r0_q * TG_R;
    tgg_q <= g0_q * TG_G;
    tgb_q <= b0_q * TG_B;

    tbr_q <= r0_q * TB_R;
    tbg_q <= g0_q * TB_G;
    tbb_q <= b0_q * TB_B;

    // clock 3: divide by 256 and add
    tr_q <= trr_q[15:8] + trg_q[15:8] + trb_q[15:8];
    tg_q <= tgr_q[15:8] + tgg_q[15:8] + tgb_q[15:8];
```

```
    tb_q <= tbr_q[15:8] + tbg_q[15:8] + tbb_q[15:8];

    // clock 4: determine outputs
    rf_q <= (tr_q < 8'd255) ? tr_q[7:0] : 8'd255;
    gf_q <= (tg_q < 8'd255) ? tg_q[7:0] : 8'd255;
    bf_q <= (tb_q < 8'd255) ? tb_q[7:0] : 8'd255;

  end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    14:44:07 12/02/2016
// Design Name:
// Module Name:    sobel_op
// Project Name:
// Target Devices:
// Tool versions:
// Description: Implements and applies a 3x3 Sobel operator to an input matrix of
//              values. Outputs a gradient value. Pipelined with 4 stages.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sobel_op(
    input clk, rst,
    input [7:0] a0, a1, a2,
    input [7:0] a7, a3,
    input [7:0] a6, a5, a4,
    output [15:0] gradient
  );

  // Sobel 3x3 operator

  // convolutional masks for Sobel operator
  //   Gx = ( -1 0 1 )   Gy = (  1  2  1 )
  //        ( -2 0 2 )        (  0  0  0 )
  //        ( -1 0 1 )        ( -1 -2 -1 )

  //  labeling of pixels
  //    a0    a1    a2
  //    a7   pix    a3
  //    a6    a5    a4

  // partial derivatives
  // gx = (a0+2a1+a2)-(a6+2a5+a4)
  // gy = (a2+2a3+a4)-(a0+2a7+a6)

  // gradient computation: G = sqrt(gx^2 + gy^2)
  // approximation: G = abs(gx) + abs(gy)

  reg [7:0] a0_q, a1_q, a2_q, a7_q, a3_q, a6_q, a5_q, a4_q;  // to latch inputs

  reg signed [15:0] gx_q, gy_q;   // partial derivatives
  reg [15:0] abs_gx_q, abs_gy_q;  // absolute values
  reg [15:0] gradient_q;

  assign gradient = gradient_q;

  always @(posedge clk) begin
    // clock 1: latch inputs
    {a0_q, a1_q, a2_q} <= {a0, a1, a2};
    {a7_q, a3_q} <= {a7, a3};
    {a6_q, a5_q, a4_q} <= {a6, a5, a4};

    // clock 2: compute partial derivatives
    gx_q <= (a0_q+(a1_q<<1)+a2_q)-(a6_q+(a5_q<<1)+a4_q);
    gy_q <= (a2_q+(a3_q<<1)+a4_q)-(a0_q+(a7_q<<1)+a6_q);

    // clock 3: compute absolute values
    abs_gx_q <= gx_q[15] ? (~gx_q)+1'b1 : gx_q;
    abs_gy_q <= gy_q[15] ? (~gy_q)+1'b1 : gy_q;

    // clock 4: compute gradient
    gradient_q <= abs_gx_q + abs_gy_q;
  end
```

```
endmodule
```

```
`define STATUS_RESET             4'h0
`define STATUS_READ_ID           4'h1
`define STATUS_CLEAR_LOCKS       4'h2
`define STATUS_ERASING           4'h3
`define STATUS_WRITING           4'h4
`define STATUS_READING           4'h5
`define STATUS_SUCCESS           4'h6
`define STATUS_BAD_MANUFACTURER  4'h7
`define STATUS_BAD_SIZE          4'h8
`define STATUS_LOCK_BIT_ERROR    4'h9
`define STATUS_ERASE_BLOCK_ERROR 4'hA
`define STATUS_WRITE_ERROR       4'hB
`define STATUS_READ_WRONG_DATA   4'hC

`define NUM_BLOCKS 128
`define BLOCK_SIZE 64*1024
`define LAST_BLOCK_ADDRESS ((`NUM_BLOCKS-1)*`BLOCK_SIZE)
`define LAST_ADDRESS (`NUM_BLOCKS*`BLOCK_SIZE-1)

`define FLASHOP_IDLE  2'b00
`define FLASHOP_READ  2'b01
`define FLASHOP_WRITE 2'b10

module test_fsm (reset, clock, fop, faddress, fwdata, frdata, fbusy, dots, mode, bus
y, datain, addrin, state);
   input reset, clock;
   output [1:0] fop;
   output [22:0] faddress;
   output [15:0] fwdata;
   input [15:0]  frdata;
   input fbusy;
   output [639:0] dots;
   input [1:0] mode;
   output busy;
   input [15:0] datain;
   input [22:0] addrin;
   output state;

   reg [1:0] fop;
   reg [22:0] faddress;
   reg [15:0] fwdata;
   reg [639:0] dots;
   reg busy;
   reg [15:0] data_to_store;

   ////////////////////////////////////////////////////////////////////////
   //
   // State Machine
   //
   ////////////////////////////////////////////////////////////////////////

   reg [7:0] state;
   reg [3:0] status;

   parameter MODE_IDLE  = 0;
   parameter MODE_INIT  = 1;
   parameter MODE_WRITE = 2;
   parameter MODE_READ  = 3;

   parameter MAX_ADDRESS = 23'h030000;

   parameter HOME = 8'h12;


   always @(posedge clock)
     if (reset)
       begin
                       state <= HOME;
                       status <= `STATUS_RESET;
                       faddress <= 0;
                       fop <= `FLASHOP_IDLE;
                       busy <= 1;
       end
```

```
      else if (!fbusy && (fop == `FLASHOP_IDLE))
        case (state)

          HOME://12
                case(mode)
                        MODE_INIT: begin
                                state <= 8'h00;
                                busy <= 1;
                        end

                        MODE_WRITE: begin
                                state <= 8'h0C;
                                busy <= 1;
                        end

                        MODE_READ: begin
                                busy <= 1;
                                if(status == `STATUS_READING)
                                        state <= 8'h11;
                                else
                                        state <= 8'h10;
                        end

                        default: begin
                                state <= HOME;
                                busy <= 0;
                        end
                endcase

          ////////////////////////////////////////////////////////////////////////
          // Wipe It
          ////////////////////////////////////////////////////////////////////////
          8'h00:
                begin
                        // Issue "read id codes" command
                        status <= `STATUS_READ_ID;
                        faddress <= 0;
                        fwdata <= 16'h0090;
                        fop <= `FLASHOP_WRITE;
                        state <= state+1;
                end

          8'h01:
            begin
               // Read manufacturer code
               faddress <= 0;
               fop <= `FLASHOP_READ;
               state <= state+1;
            end

          8'h02:
            if (frdata != 16'h0089) // 16'h0089 = Intel
              status <= `STATUS_BAD_MANUFACTURER;
            else
              begin
                 // Read the device size code
                 faddress <= 1;
                 fop <= `FLASHOP_READ;
                 state <= state+1;
              end

          8'h03:
            if (frdata != 16'h0018) // 16'h0018 = 128Mbit
              status <= `STATUS_BAD_SIZE;
            else
              begin
                 faddress <= 0;
                 fwdata <= 16'hFF;
                 fop <= `FLASHOP_WRITE;
                 state <= state+1;
              end

          8'h04:
```

```
            begin
               // Issue "clear lock bits" command
               status <= `STATUS_CLEAR_LOCKS;
               faddress <= 0;
               fwdata <= 16'h60;
               fop <= `FLASHOP_WRITE;
               state <= state+1;
            end

          8'h05:
            begin
               // Issue "confirm clear lock bits" command
               faddress <= 0;
               fwdata <= 16'hD0;
               fop <= `FLASHOP_WRITE;
               state <= state+1;
            end

          8'h06:
            begin
               // Read status
               faddress <= 0;
               fop <= `FLASHOP_READ;
               state <= state+1;
            end

          8'h07:
            if (frdata[7] == 1) // Done clearing lock bits
              if (frdata[6:1] == 0) // No errors
                begin
                   faddress <= 0;
                   fop <= `FLASHOP_IDLE;
                   state <= state+1;
                end
              else
                status <= `STATUS_LOCK_BIT_ERROR;
            else // Still busy, reread status register
              begin
                 faddress <= 0;
                 fop <= `FLASHOP_READ;
              end

          ////////////////////////////////////////////////////////////////////////
          // Block Erase Sequence
          ////////////////////////////////////////////////////////////////////////
          8'h08:
            begin
               status <= `STATUS_ERASING;
               fwdata <= 16'h20; // Issue "erase block" command
               fop <= `FLASHOP_WRITE;
               state <= state+1;
            end

          8'h09:
            begin
               fwdata <= 16'hD0; // Issue "confirm erase" command
               fop <= `FLASHOP_WRITE;
               state <= state+1;
            end
          8'h0A:
            begin
               fop <= `FLASHOP_READ;
               state <= state+1;
            end
          8'h0B:
            if (frdata[7] == 1) // Done erasing block
              if (frdata[6:1] == 0) // No errors
                if (faddress != MAX_ADDRESS) // `LAST_BLOCK_ADDRESS)
                  begin
                     faddress <= faddress+`BLOCK_SIZE;
                     fop <= `FLASHOP_IDLE;
                     state <= state-3;
                  end
```

```
            else
              begin
                 faddress <= 0;
                 fop <= `FLASHOP_IDLE;
                 state <= HOME;              //done erasing, go home
              end
          else // Erase error detected
            status <= `STATUS_ERASE_BLOCK_ERROR;
        else // Still busy
          fop <= `FLASHOP_READ;

      ////////////////////////////////////////////////////////////////////
      // Write Mode
      ////////////////////////////////////////////////////////////////////
      8'h0C:
             begin
                     data_to_store <= datain;
                     status <= `STATUS_WRITING;
                     fwdata <= 16'h40; // Issue "setup write" command
                     fop <= `FLASHOP_WRITE;
                     state <= state+1;
             end

      8'h0D:
        begin
           fwdata <= data_to_store; // Finish write
           fop <= `FLASHOP_WRITE;
           state <= state+1;
        end
      8'h0E:
        begin
           // Read status register
           fop <= `FLASHOP_READ;
           state <= state+1;
        end
      8'h0F:
        if (frdata[7] == 1) // Done writing
          if (frdata[6:1] == 0) // No errors
            if (faddress != 23'h7FFFFF) // `LAST_ADDRESS)
                              begin
                                      faddress <= faddress+1;
                                      fop <= `FLASHOP_IDLE;
                                      state <= HOME;
                              end
                      else
                              status <= `STATUS_WRITE_ERROR;
          else // Write error detected
            status <= `STATUS_WRITE_ERROR;
        else // Still busy
          fop <= `FLASHOP_READ;

      ////////////////////////////////////////////////////////////////////
      // Read Mode INIT
      ////////////////////////////////////////////////////////////////////
      8'h10:
        begin
           status <= `STATUS_READING;
           fwdata <= 16'hFF; // Issue "read array" command
           fop <= `FLASHOP_WRITE;
           faddress <= 0;
           state <= state+1;
        end

      ////////////////////////////////////////////////////////////////////
      // Read Mode
      ////////////////////////////////////////////////////////////////////
      8'h11:
        begin
              faddress <= addrin;
           fop <= `FLASHOP_READ;
           state <= HOME;
        end
```

```
        default:
                begin
                        status <= `STATUS_BAD_MANUFACTURER;
                        faddress <= 0;
                        state <= HOME;
                end

        endcase
      else
        fop <= `FLASHOP_IDLE;

   function [39:0] nib2char;
      input [3:0] nib;
      begin
         case (nib)
           4'h0: nib2char = 40'b00111110_01010001_01001001_01000101_00111110;
           4'h1: nib2char = 40'b00000000_01000010_01111111_01000000_00000000;
           4'h2: nib2char = 40'b01100010_01010001_01001001_01001001_01000110;
           4'h3: nib2char = 40'b00100010_01000001_01001001_01001001_00110110;
           4'h4: nib2char = 40'b00011000_00010100_00010010_01111111_00010000;
           4'h5: nib2char = 40'b00100111_01000101_01000101_01000101_00111001;
           4'h6: nib2char = 40'b00111100_01001010_01001001_01001001_00110000;
           4'h7: nib2char = 40'b00000001_01110001_00001001_00000101_00000011;
           4'h8: nib2char = 40'b00110110_01001001_01001001_01001001_00110110;
           4'h9: nib2char = 40'b00000110_01001001_01001001_00101001_00011110;
           4'hA: nib2char = 40'b01111110_00001001_00001001_00001001_01111110;
           4'hB: nib2char = 40'b01111111_01001001_01001001_01001001_00110110;
           4'hC: nib2char = 40'b00111110_01000001_01000001_01000001_00100010;
           4'hD: nib2char = 40'b01111111_01000001_01000001_01000001_00111110;
           4'hE: nib2char = 40'b01111111_01001001_01001001_01001001_01000001;
           4'hF: nib2char = 40'b01111111_00001001_00001001_00001001_00000001;
         endcase
      end
   endfunction

   wire [159:0] data_dots;
   assign data_dots = {nib2char(frdata[15:12]), nib2char(frdata[11:8]),
                       nib2char(frdata[7:4]), nib2char(frdata[3:0])};

   wire [239:0] address_dots;
   assign address_dots = {nib2char({ 1'b0, faddress[22:20]}),
                          nib2char(faddress[19:16]),
                          nib2char(faddress[15:12]),
                          nib2char(faddress[11:8]),
                          nib2char(faddress[7:4]),
                          nib2char(faddress[3:0])};

   always @(status or address_dots or data_dots)
     case (status)
       `STATUS_RESET:
         dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                  40'b01111111_01001001_01001001_01001001_01000001, // E
                  40'b00100110_01001001_01001001_01001001_00110010, // S
                  40'b01111111_01001001_01001001_01001001_01000001, // E
                  40'b00000001_00000001_01111111_00000001_00000001, // T
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00001000_00001000_00001000_00001000_00001000, // -
                  40'b00001000_00001000_00001000_00001000_00001000, // -
                  40'b00001000_00001000_00001000_00001000_00001000, // -
                  40'b00001000_00001000_00001000_00001000_00001000, // -
                  40'b00001000_00001000_00001000_00001000_00001000, // -
                  40'b00001000_00001000_00001000_00001000_00001000};// -
       `STATUS_READ_ID:
         dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                  40'b01111111_01001001_01001001_01001001_01000001, // E
                  40'b01111110_00001001_00001001_00001001_01111110, // A
                  40'b01111111_01000001_01000001_01000001_00111110, // D
                  40'b00000000_00000000_00000000_00000000_00000000, //
                  40'b00000000_01000001_01111111_01000001_00000000, // I
```

```
                40'b01111111_01000001_01000001_01000001_00111110, // D
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                address_dots};
        `STATUS_CLEAR_LOCKS:
          dots <= {40'b00111110_01000001_01000001_01000001_00100010, // C
                40'b01111111_01000000_01000000_01000000_01000000, // L
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_01000000_01000000_01000000_01000000, // L
                40'b00111110_01000001_01000001_01000001_00111110, // O
                40'b00111110_01000001_01000001_01000001_00100010, // C
                40'b01111111_00001000_00010100_00100010_01000001, // K
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b00000000_00000000_00000000_00000000_00000000, //
                address_dots};
        `STATUS_ERASING:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111110_00001001_00001001_00001001_01111110, // A
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b01111111_00000010_00000100_00001000_01111111, // N
                40'b00111110_01000001_01001001_01001001_00111010, // G
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                address_dots};
        `STATUS_WRITING:
          dots <= {40'b01111111_00100000_00011000_00100000_01111111, // W
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b00000001_00000001_01111111_00000001_00000001, // T
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b01111111_00000010_00000100_00001000_01111111, // N
                40'b00111110_01000001_01001001_01001001_00111010, // G
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                address_dots};
        `STATUS_READING:
          dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111110_00001001_00001001_00001001_01111110, // A
                40'b01111111_01000001_01000001_01000001_00111110, // D
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b01111111_00000010_00000100_00001000_01111111, // N
                40'b00111110_01000001_01001001_01001001_00111010, // G
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                address_dots};
        `STATUS_SUCCESS:
          dots <= {40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_00001001_00001001_00001001_00000110, // P
                40'b01111110_00001001_00001001_00001001_01111110, // A
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_01000001_01000001_01000001_00111110, // D
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00101010_00011100_01111111_00011100_00101010, // *
                40'b00000000_00000000_00000000_00000000_00000000};//
        `STATUS_BAD_MANUFACTURER:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
```

```
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_00000010_00001100_00000010_01111111, // M
                40'b01111110_00001001_00001001_00001001_01111110, // A
                40'b01111111_00000010_00000100_00001000_01111111, // N
                40'b01111111_00001001_00001001_00001001_00000001, // U
                40'b01111111_00001001_00001001_00001001_00000001, // F
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00000000_00000000_00000000_00000000_00000000, //
                data_dots};
        `STATUS_BAD_SIZE:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b01100001_01010001_01001001_01000101_01000011, // Z
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b00000000_00000000_00000000_00000000_00000000,
                40'b00000000_00000000_00000000_00000000_00000000,
                40'b00000000_00000000_00000000_00000000_00000000,
                data_dots};
        `STATUS_LOCK_BIT_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_01000000_01000000_01000000_01000000, // L
                40'b00111110_01000001_01000001_01000001_00111110, // O
                40'b00111110_01000001_01000001_01000001_00100010, // C
                40'b01111111_00001000_00010100_00100010_01000001, // K
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b00000000_00000000_00000000_00000000_00000000,
                40'b00000000_00000000_00000000_00000000_00000000,
                data_dots};
        `STATUS_ERASE_BLOCK_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111110_00001001_00001001_00001001_01111110, // A
                40'b00100110_01001001_01001001_01001001_00110010, // S
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b00000000_00000000_00000000_00000000_00000000,
                40'b00000000_00000000_00000000_00000000_00000000,
                data_dots};
        `STATUS_WRITE_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000, //
                40'b01111111_00100000_00011000_00100000_01111111, // W
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_01000001_01111111_01000001_00000000, // I
                40'b00000001_00000001_01111111_00000001_00000001, // T
                40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b00000000_00000000_00000000_00000000_00000000,
                40'b00000000_00000000_00000000_00000000_00000000,
                data_dots};
        `STATUS_READ_WRONG_DATA:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b01111111_00001001_00011001_00101001_01000110, // R
                40'b00000000_00110110_00110110_00000000_00000000, // :
                40'b00000000_00000000_00000000_00000000_00000000,
```

```
                    address_dots,
                    40'b00000000_00000000_00000000_00000000_00000000,
                    data_dots};
            default:
            dots <= {16{40'b01010101_00101010_01010101_00101010_01010101}};
        endcase

endmodule
```

```
`timescale 1ns / 1ps
module stringmaker #(parameter STRING_LENGTH=9)(input clk,
                                    input [10:0] x,hcount,
                                    input [9:0] y,vcount,
                                    input [1:0] background,
                                    input [5:0] numchar,
                                    input ready,
                                    input [STRING_LENGTH*8-1:0]string,
                                    input custom,
                                    output[23:0] pixel);

        parameter ARRAY_LEN=8*STRING_LENGTH;
        parameter leftmax=1023;
        parameter bottommax=759;

        reg [7:0] letter=8'h57;
        wire [10:0] xpos;
        reg [10:0] xshift;
        reg hcountold;
        reg vcountold;
        reg [10:0] linecount=1;
        reg [10:0] condition;
        reg [5:0]countershift;
        reg [8:0]letterfind;
        reg [9:0]letterfindnext;
        reg valid;
        reg [5:0] numcounter;
        reg [ARRAY_LEN-1:0]stringhold;

        always @(posedge clk) begin

                //letter generator
                if(custom)begin //custom text
                        if (ready) stringhold<=string;
                                hcountold<=hcount[0];
                                vcountold<=vcount[0];
                                if ((hcount[0]!=hcountold)&&!(vcount[0]!=vcountold))
 linecount<=1+linecount;
                                if (numchar==0) letter<=8'h20; //set letter to space
 to clean out in a sense
                        if (vcount[0]!=vcountold&&numchar>0) begin//reset and first
letter only if there is
                                        linecount<=0;
                                        letter<=stringhold[ARRAY_LEN-1:ARRAY_LEN-8];
                                        xshift<=0;
                                        countershift<=1;
                                        letterfind<=8;
                                        condition<=18; //18 pixels is an aesthetical
ly pleasing offset
                                        numcounter<=numchar-1;
                                        end
                                if ((linecount==condition+x)&&numcounter!=0&&numchar
>=1) begin //each loop it will move 18 pixels over
                                        numcounter<=numcounter-1;
                                        if (countershift+2<numchar)countershift<=1+c
ountershift;
                                        //all look ahead multiplications
                                        condition<=18*(countershift+1); //look ahead
 one because it will be the condition for the next letter
                                        letterfind<=(countershift+1)*8; //look ahead
 one becuase it will be used to find the next letter which is inherently one larger
than countershift
                                        letterfindnext<=(countershift+2)*8;
                                        xshift<=(condition+3);
                                        if ((letterfind-8)<ARRAY_LEN)begin
                                                letter<={stringhold[ARRAY_LEN-(1+let
terfind)],
                                                                        stringhold[A
RRAY_LEN-(2+letterfind)],
                                                                        stringhold[A
RRAY_LEN-(3+letterfind)],
                                                                        stringhold[A
RRAY_LEN-(4+letterfind)],
```

```
                                                                stringhold[A
RRAY_LEN-(5+letterfind)],
                                                                stringhold[A
RRAY_LEN-(6+letterfind)],
                                                                stringhold[A
RRAY_LEN-(7+letterfind)],
                                                                stringhold[A
RRAY_LEN-(8+letterfind)]};
                                                        end
                                        end
                                end
                        else begin // all the default text code.
                                hcountold<=hcount[0];
                                vcountold<=vcount[0];
                                if ((hcount[0]!=hcountold)&&!(vcount[0]!=vcountold)) linecou
nt<=1+linecount;
                                if (vcount[0]!=vcountold) begin//reset and G
                                        linecount<=0;
                                        letter<=8'h47;
                                        xshift<=0;
                                        end
                                if (linecount==18+x) begin//R
                                        xshift<=20;
                                        letter<=8'h52;
                                end
                                if (linecount==36+x) begin //E
                                        xshift<=38;
                                        letter<=8'h45;
                                        end
                                if (linecount==54+x) begin //E
                                        xshift<=56;
                                        letter<=8'h45;
                                        end
                                if (linecount==72+x) begin //T
                                        xshift<=74;
                                        letter<=8'h54;
                                        end
                                if (linecount==90+x) begin //I
                                        xshift<=92;
                                        letter<=8'h49;
                                        end
                                if (linecount==108+x) begin //N
                                        xshift<=111;
                                        letter<=8'h4E;
                                        end
                                if (linecount==126+x) begin //G
                                        xshift<=129;
                                        letter<=8'h47;
                                        end
                                if (linecount==144+x) begin //S
                                        xshift<=146;
                                        letter<=8'h53;
                                        end
                                if (linecount==168+x) begin //F (extra for the space
                                        xshift<=170;
                                        letter<=8'h46;
                                        end
                                if (linecount==186+x) begin //R
                                        xshift<=188;
                                        letter<=8'h52;
                                        end
                                if (linecount==204+x) begin //O
                                        xshift<=206;
                                        letter<=8'h4F;
                                        end
                                if (linecount==222+x) begin //M
                                        xshift<=224;
                                        letter<=8'h4D;
                                        end
                                case(background)
                                        0:begin//Paris
                                                if (linecount==246+x) begin //P
                                                        xshift<=248;
```

```
                letter<=8'h50;
                end
        if (linecount==264+x) begin //A
                xshift<=266;
                letter<=8'h41;
                end
        if (linecount==282+x) begin //R
                xshift<=284;
                letter<=8'h52;
                end
        if (linecount==300+x) begin //I
                xshift<=302;
                letter<=8'h49;
                end
        if (linecount==318+x) begin //S
                xshift<=321;
                letter<=8'h53;
                end
        end
1:begin//Rome
        if (linecount==246+x) begin //R
                xshift<=248;
                letter<=8'h52;
                end
        if (linecount==264+x) begin //O
                xshift<=266;
                letter<=8'h4F;
                end
        if (linecount==282+x) begin //M
                xshift<=284;
                letter<=8'h4D;
                end
        if (linecount==300+x) begin //E
                xshift<=302;
                letter<=8'h45;
                end
        end

2:begin//(the)Amazon
        if (linecount==245+x) begin //T
                xshift<=248;
                letter<=8'h54;
                end
        if (linecount==264+x) begin //H
                xshift<=266;
                letter<=8'h48;
                end
        if (linecount==282+x) begin //E
                xshift<=284;
                letter<=8'h45;
                end
        if (linecount==306+x) begin //A ADD SPACE
                xshift<=308;
                letter<=8'h41;
                end
        if (linecount==324+x) begin //M
                xshift<=326;
                letter<=8'h4D;
                end
        if (linecount==342+x) begin //A
                xshift<=346;
                letter<=8'h41;
                end
        if (linecount==360+x) begin //Z
                xshift<=362;
                letter<=8'h5A;
                end
        if (linecount==378+x) begin //O
                xshift<=380;
                letter<=8'h4F;
                end
        if (linecount==396+x) begin //N
                xshift<=398;
```

```
                                        letter<=8'h4E;
                                        end

                                end
                        3:begin//London
                                if (linecount==246+x) begin //L
                                        xshift<=248;
                                        letter<=8'h4C;
                                        end
                                if (linecount==264+x) begin //O
                                        xshift<=266;
                                        letter<=8'h4F;
                                        end
                                if (linecount==282+x) begin //N
                                        xshift<=284;
                                        letter<=8'h4E;
                                        end
                                if (linecount==300+x) begin //D
                                        xshift<=302;
                                        letter<=8'h44;
                                        end
                                if (linecount==318+x) begin //O
                                        xshift<=320;
                                        letter<=8'h4F;
                                        end
                                if (linecount==336+x) begin //N
                                        xshift<=338;
                                        letter<=8'h4E;
                                        end
                                end
                        endcase
                end
        end
        assign xpos=x+xshift;
        textcaller lettermaker(clk,xpos,hcount,y,vcount,letter,pixel);
endmodule

module textcaller #(parameter WIDTH = 15,HEIGHT =17)
        (input pixel_clk,
         input [10:0] x,hcount,
         input [9:0] y,vcount,
     input [7:0] letter,
     output reg[23:0] pixel);

        reg [8:0] image_addr;
        wire [8:0] image_mem;
    wire [14:0] image_bits;
    reg [3:0] mem_iter=0;
        reg [3:0] hdelay=0;
        reg hcountold;
        reg vcountold;

   always @(posedge pixel_clk) begin
                        case(letter)
                                8'h41:image_addr<=0; //A
                                8'h42:image_addr<=17;//B
                                8'h43:image_addr<=34;//C
                                8'h44:image_addr<=51;//D
                                8'h45:image_addr<=68;//E
                                8'h46:image_addr<=85;//F
                                8'h47:image_addr<=102;//G
                                8'h48:image_addr<=119;//H
                                8'h49:image_addr<=136;//I
                                8'h4A:image_addr<=153;//J
                                8'h4B:image_addr<=170;//K
                                8'h4C:image_addr<=187;//L
                                8'h4D:image_addr<=204;//M
                                8'h4E:image_addr<=221;//N
                                8'h4F:image_addr<=238;//O
                                8'h50:image_addr<=255;//P
                                8'h51:image_addr<=272;//Q
                                8'h52:image_addr<=289;//R
                                8'h53:image_addr<=306;//S
```

```
                                8'h54:image_addr<=323;//T
                                8'h55:image_addr<=340;//U
                                8'h56:image_addr<=357;//V
                                8'h57:image_addr<=374;//W
                                8'h58:image_addr<=391;//X
                                8'h59:image_addr<=408;//Y
                                8'h5A:image_addr<=425;//Z
                                8'h20:image_addr<=442;//SPACE
                               default: image_addr<=442;//
                        endcase

                        hcountold<=hcount[0];
                        vcountold<=vcount[0];

                        if (vcount==y) mem_iter<=0;

                        //if we are within range and vcount has changed. then go get
 the next line in memory
                        if ((~(vcount==y))&&(vcount[0]!=vcountold)&&(vcount>y)&&(vco
unt<HEIGHT+y)) mem_iter<=mem_iter+1;

                        //if we are within range and hcount has changed. then look a
t the next pixel
                        if ((hcount[0]!=hcountold)&&(hcount>=x)&&(hcount<=WIDTH+x))
hdelay<=hdelay+1;
                        if (image_bits[15-hdelay[3:0]]==1)pixel <= 24'hFF0000; //che
ck for 0 or 1. assign color accourdingly
                        else pixel <= 24'hFFFFFF;
                end
        assign image_mem=image_addr+mem_iter; //memory offset to select correct lett
er plus the iterator
        font_rom alphabet(pixel_clk,image_mem,image_bits);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Diana Wofk
//
// Create Date:    14:44:01 11/26/2016
// Design Name:
// Module Name:    uart_tx
// Project Name:
// Target Devices:
// Tool versions:
// Description: Simple parameterized UART TX module. Default baudrate is 115200.
//              Default TX settings: 1 start bit, 8 data bits, 2 stop bits. This
//              implementation does not support RTS/CTS flow control.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module uart_tx #(
    parameter BAUD_RATE = 115200,
    parameter CLK_IN = 40000000,     // 40MHz input clock
    parameter START_BITS = 1,
    parameter DATA_BITS = 8,
    parameter STOP_BITS = 2
  )(
    input clk, rst,           // 40 MHz input clk
    input tx_en,              // raise to transmit
    input [7:0] data_in,     // 8-bit data input
    output bit_out,           // UART transmit wire
    output tx_busy,           // asserted when transmitting
    output [3:0] bit_ctr     // bit counter for transmitting bytes
  );

  localparam BITCOUNT = START_BITS + DATA_BITS + STOP_BITS;

  reg bit_out_q;  // data bit being sent out on wire
  reg [3:0] bit_ctr_q;  // counter for shifting data bits
  reg [(BITCOUNT-1):0] tx_shifter_q;  // acts as a data buffer

  // output assignments
  assign tx_busy = bit_ctr > 0;
  assign bit_ctr = bit_ctr_q;
  assign bit_out = bit_out_q;

  // ***********************************************
  // TX clock Generation (for 115200 baud)
  // ***********************************************

  reg [25:0] tx_clk_ctr_q = 0;  // counter used in generating tx_clk
  wire tx_clk = ((CLK_IN-tx_clk_ctr_q) < BAUD_RATE);  // for serial transmission

  always @(posedge clk)
    tx_clk_ctr_q <= ((CLK_IN-tx_clk_ctr_q) < BAUD_RATE) ? 0 : tx_clk_ctr_q+BAUD_RATE
;

  // ********************************
  // Data Transmission
  // ********************************

  always @(posedge clk) begin
    if (rst) begin
      bit_out_q <= 1'b1;
      bit_ctr_q <= 0;
      tx_shifter_q <= 0;
    end else begin

      // there is a byte available for transmission
      if (tx_en & (bit_ctr_q <= 1)) begin
```

```
        bit_out_q <= 1'b1; // can be interpreted as a stop bit if necessary
        bit_ctr_q <= BITCOUNT;  // num of bits to be transmitted for this byte
        tx_shifter_q <= {{STOP_BITS{1'b1}}, data_in[7:0], {START_BITS{1'b0}}};
      end

      // transmit the start/data/stop bits
      if (tx_clk && (bit_ctr_q > 1)) begin
        bit_out_q <= tx_shifter_q[0];
        bit_ctr_q <= bit_ctr_q-1'b1;
        tx_shifter_q <= {1'b1, tx_shifter_q[(BITCOUNT-1):1]};
      end

    end
  end

endmodule
```

```
//reads data and puts it on out
module usb_input(clk,reset,data,rd,rxf,out,newout,hold,state);
        input clk, reset;       //clock and reset
        input [7:0] data;       //the data pins from the USB fifo
        input rxf;                      //the rxf pin from the USB fifo
        output rd;                      //the rd pin from the USB fifo
        reg rd;

        output[7:0] out;        //this is where data goes when it has been read from
 the fifo
        reg[7:0] out;
        output newout;          //when this is high, out contains a new chunk of dat
a
        reg newout;
        input hold;                     //as long as hold is high, this module sits
                                        //still module and will not accept n
ew data from the fifo

        output state;           //for debugging purposes
        reg[3:0] state;

        parameter RESET                 = 0;            //state data
        parameter WAIT                  = 1;
        parameter WAIT2                 = 2;
        parameter WAIT3                 = 3;
        parameter DATA_COMING   = 4;
        parameter DATA_COMING_2 = 5;
        parameter DATA_COMING_3 = 6;
        parameter DATA_COMING_4 = 7;
        parameter DATA_COMING_5 = 8;
        parameter DATA_HERE     = 9;
        parameter DATA_LEAVING  =10;
        parameter DATA_LEAVING_2=11;
        parameter DATA_LEAVING_3=12;

        initial
                state <= WAIT;

        always @ (posedge clk)
                if(reset)
                        begin
                                newout <= 0;
                                rd <= 1;                        //we can't read data
                                state <= WAIT;
                        end
                else
                        if(~hold)
                                begin
                                        newout <= 0;
                                        case(state)
                                        WAIT:
                                                if(~rxf)                //if rxf is
low and nobody's asking us to wait then there is data waiting for us
                                                        begin
                                                                rd <= 1;
                                //so ask for it
                                                                state <= WAIT2; //an
d start waiting for it
                                                        end

                                        WAIT2:
                                                if(~rxf)                //double che
ck
                                                        begin
                                                                rd <= 1;

                                                                state <= WAIT3;
                                                        end
                                                else
                                                        state <= WAIT;

                                        WAIT3:
                                                if(~rxf)                //and triple
```

```
 check (should only need one, but oh well...)
                                                        begin
                                                                rd <= 0;

                                                                state <= DATA_COMING
;
                                                        end
                                                else
                                                        state <= WAIT;

                                        DATA_COMING:            //once rd goes low w
e gotta wait a bit for the data to stabilize
                                                state <= DATA_COMING_2;

                                        DATA_COMING_2:
                                                state <= DATA_COMING_3;

                                        DATA_COMING_3:
                                                state <= DATA_HERE;

                                        DATA_HERE:
                                                begin
                                                        out <= data;    //the data i
s valid by now so read it
                                                        state <= DATA_LEAVING;
                                                        newout <= 1;    //let folks
know we've got new data
                                                end

                                        DATA_LEAVING:                   //wait a cyc
le to clear the data to make sure we latch onto it correctly
                                                begin
                                                        rd <= 1;
                                                        state <= DATA_LEAVING_2;
                                                        newout <= 0;    //let folks
know the data's a clock cycle old now
                                                end

                                        DATA_LEAVING_2:         //wait another cycle
 to make sure that the RD to RD pre-charge time is met
                                                state <= DATA_LEAVING_3;


                                        DATA_LEAVING_3:         //wait another cycle
 to make sure that the RD to RD pre-charge time is met
                                                state <= WAIT;


                                        default:
                                                state <= WAIT;
                                endcase
                        end
endmodule
```

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

//////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
   output [29:0] ycrcb;
   output        f;
   output        v;
   output        h;
   output        data_valid;
   // output [4:0] state;

   parameter     SYNC_1 = 0;
   parameter     SYNC_2 = 1;
   parameter     SYNC_3 = 2;
   parameter     SAV_f1_cb0 = 3;
   parameter     SAV_f1_y0 = 4;
   parameter     SAV_f1_cr1 = 5;
   parameter     SAV_f1_y1 = 6;
   parameter     EAV_f1 = 7;
   parameter     SAV_VBI_f1 = 8;
   parameter     EAV_VBI_f1 = 9;
   parameter     SAV_f2_cb0 = 10;
   parameter     SAV_f2_y0 = 11;
   parameter     SAV_f2_cr1 = 12;
   parameter     SAV_f2_y1 = 13;
   parameter     EAV_f2 = 14;
   parameter     SAV_VBI_f2 = 15;
   parameter     EAV_VBI_f2 = 16;



   // In the start state, the module doesn't know where
   // in the sequence of pixels, it is looking.

   // Once we determine where to start, the FSM goes through a normal
   // sequence of SAV process_YCrCb EAV... repeat

   // The data stream looks as follows
   // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV
 sequence
   // There are two things we need to do:
```

```
   //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
   //   2. Decode the subsequent data

   reg [4:0]     current_state = 5'h00;
   reg [9:0]     y = 10'h000;  // luminance
   reg [9:0]     cr = 10'h000; // chrominance
   reg [9:0]     cb = 10'h000; // more chrominance

   assign        state = current_state;

   always @ (posedge clk)
     begin
        if (reset)
          begin

          end
        else
          begin
             // these states don't do much except allow us to know where we are in t
he stream.
             // whenever the synchronization code is seen, go back to the sync_state
 before
             // transitioning to the new state
             case (current_state)
               SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
               SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
               SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                        (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                                        (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                        (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                        (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                        (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                                        (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                        (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC
_1;

               SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_
f1_y0;
               SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f
1_cr1;
               SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_
f1_y1;
               SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f
1_cb0;

               SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_
f2_y0;
               SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f
2_cr1;
               SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_
f2_y1;
               SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f
2_cb0;

               // These states are here in the event that we want to cover these sig
nals
               // in the future. For now, they just send the state machine back to S
YNC_1
               EAV_f1: current_state <= SYNC_1;
               SAV_VBI_f1: current_state <= SYNC_1;
               EAV_VBI_f1: current_state <= SYNC_1;
               EAV_f2: current_state <= SYNC_1;
               SAV_VBI_f2: current_state <= SYNC_1;
               EAV_VBI_f2: current_state <= SYNC_1;

             endcase
          end
     end // always @ (posedge clk)

   // implement our decoding mechanism

   wire y_enable;
   wire cr_enable;
```

```
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
   assign y_enable = (current_state == SAV_f1_y0) ||
                     (current_state == SAV_f1_y1) ||
                     (current_state == SAV_f2_y0) ||
                     (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg    f = 0;

   always @ (posedge clk)
     begin
        y <= y_enable ? tv_in_ycrcb : y;
        cr <= cr_enable ? tv_in_ycrcb : cr;
        cb <= cb_enable ? tv_in_ycrcb : cb;
        f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
     end

endmodule



//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
// Register 0
//////////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                              4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

//////////////////////////////////////////////////////////////////////////////////
// Register 1
//////////////////////////////////////////////////////////////////////////////////
```

```
`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM, `FOUR_TIMES_S
AMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////
// Register 2
////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT, `
INTERFACE_SELECT}
```

```
////////////////////////////////////////////////////////////////////////////
// Register 4
////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE, `G
PO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////
// Register 7
////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET, `FIFO_RESET
, `FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                     8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
```

```
////////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                     8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                            8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                        1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE              1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                             6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFA
ULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                            4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                            4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE                  1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL                 2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                    4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE, `TEMPORAL_DECIMATION_CO
NTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                          2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY                  1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                        1'b0
  // 0: Reference is functional
```

```
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP, `POWER
_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY, `POWER_SAVE
_CONTROL}

////////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE, `AVERAGE_BIRIGHT
NESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
```

```
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
      $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
      $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
      $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
      $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
      $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
      $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
      $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
      $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
   end

   //
   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
   //

   reg [7:0] clk_div_count, reset_count;
   reg clock_slow;
   wire reset_slow;

   initial
     begin
        clk_div_count <= 8'h00;
        // synthesis attribute init of clk_div_count is "00";
        clock_slow <= 1'b0;
        // synthesis attribute init of clock_slow is "0";
     end

   always @(posedge clock_27mhz)
     if (clk_div_count == 26)
       begin
          clock_slow <= ~clock_slow;
          clk_div_count <= 0;
       end
```

```
    else
      clk_div_count <= clk_div_count+1;

  always @(posedge clock_27mhz)
    if (reset)
      reset_count <= 100;
    else
      reset_count <= (reset_count==0) ? 0 : reset_count-1;

  assign reset_slow = reset_count != 0;

  //
  // I2C driver
  //

  reg load;
  reg [7:0] data;
  wire ack, idle;

  i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
          .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
          .sda(tv_in_i2c_data));

  //
  // State machine
  //

  reg [7:0] state;
  reg tv_in_reset_b;
  reg old_source;

  always @(posedge clock_slow)
     if (reset_slow)
       begin
          state <= 0;
          load <= 0;
          tv_in_reset_b <= 0;
          old_source <= 0;
       end
     else
       case (state)
         8'h00:
           begin
              // Assert reset
              load <= 1'b0;
              tv_in_reset_b <= 1'b0;
              if (!ack)
                state <= state+1;
           end
         8'h01:
           state <= state+1;
         8'h02:
           begin
              // Release reset
              tv_in_reset_b <= 1'b1;
              state <= state+1;
                   end
         8'h03:
           begin
              // Send ADV7185 address
              data <= 8'h8A;
              load <= 1'b1;
              if (ack)
                state <= state+1;
           end
         8'h04:
           begin
              // Send subaddress of first register
              data <= 8'h00;
              if (ack)
                state <= state+1;
           end
         8'h05:
```

```
            begin
               // Write to register 0
               data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
               if (ack)
                 state <= state+1;
            end
          8'h06:
            begin
               // Write to register 1
               data <= `ADV7185_REGISTER_1;
               if (ack)
                 state <= state+1;
            end
          8'h07:
            begin
               // Write to register 2
               data <= `ADV7185_REGISTER_2;
               if (ack)
                 state <= state+1;
            end
          8'h08:
            begin
               // Write to register 3
               data <= `ADV7185_REGISTER_3;
               if (ack)
                 state <= state+1;
            end
          8'h09:
            begin
               // Write to register 4
               data <= `ADV7185_REGISTER_4;
               if (ack)
                 state <= state+1;
            end
          8'h0A:
            begin
               // Write to register 5
               data <= `ADV7185_REGISTER_5;
               if (ack)
                 state <= state+1;
            end
          8'h0B:
            begin
               // Write to register 6
               data <= 8'h00; // Reserved register, write all zeros
               if (ack)
                 state <= state+1;
            end
          8'h0C:
            begin
               // Write to register 7
               data <= `ADV7185_REGISTER_7;
               if (ack)
                 state <= state+1;
            end
          8'h0D:
            begin
               // Write to register 8
               data <= `ADV7185_REGISTER_8;
               if (ack)
                 state <= state+1;
            end
          8'h0E:
            begin
               // Write to register 9
               data <= `ADV7185_REGISTER_9;
               if (ack)
                 state <= state+1;
            end
          8'h0F: begin
             // Write to register A
             data <= `ADV7185_REGISTER_A;
           if (ack)
```

```
            state <= state+1;
        end
        8'h10:
          begin
            // Write to register B
            data <= `ADV7185_REGISTER_B;
            if (ack)
              state <= state+1;
          end
        8'h11:
          begin
            // Write to register C
            data <= `ADV7185_REGISTER_C;
            if (ack)
              state <= state+1;
          end
        8'h12:
          begin
            // Write to register D
            data <= `ADV7185_REGISTER_D;
            if (ack)
              state <= state+1;
          end
        8'h13:
          begin
            // Write to register E
            data <= `ADV7185_REGISTER_E;
            if (ack)
              state <= state+1;
          end
        8'h14:
          begin
            // Write to register F
            data <= `ADV7185_REGISTER_F;
            if (ack)
              state <= state+1;
          end
        8'h15:
          begin
            // Wait for I2C transmitter to finish
            load <= 1'b0;
            if (idle)
              state <= state+1;
          end
        8'h16:
          begin
            // Write address
            data <= 8'h8A;
            load <= 1'b1;
            if (ack)
              state <= state+1;
          end
        8'h17:
          begin
            data <= 8'h33;
            if (ack)
              state <= state+1;
          end
        8'h18:
          begin
            data <= `ADV7185_REGISTER_33;
            if (ack)
              state <= state+1;
          end
        8'h19:
          begin
            load <= 1'b0;
            if (idle)
              state <= state+1;
          end

        8'h1A: begin
           data <= 8'h8A;
```

```
            load <= 1'b1;
            if (ack)
              state <= state+1;
          end
          8'h1B:
            begin
               data <= 8'h33;
               if (ack)
                 state <= state+1;
            end
          8'h1C:
            begin
               load <= 1'b0;
               if (idle)
                 state <= state+1;
            end
          8'h1D:
            begin
               load <= 1'b1;
               data <= 8'h8B;
               if (ack)
                 state <= state+1;
            end
          8'h1E:
            begin
               data <= 8'hFF;
               if (ack)
                 state <= state+1;
            end
          8'h1F:
            begin
               load <= 1'b0;
               if (idle)
                 state <= state+1;
            end
          8'h20:
            begin
               // Idle
               if (old_source != source) state <= state+1;
               old_source <= source;
            end
          8'h21: begin
             // Send ADV7185 address
             data <= 8'h8A;
             load <= 1'b1;
             if (ack) state <= state+1;
          end
          8'h22: begin
             // Send subaddress of register 0
             data <= 8'h00;
             if (ack) state <= state+1;
          end
          8'h23: begin
             // Write to register 0
             data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
             if (ack) state <= state+1;
          end
          8'h24: begin
             // Wait for I2C transmitter to finish
             load <= 1'b0;
             if (idle) state <= 8'h20;
          end
        endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
```

```
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;

   assign sda = sdai ? 1'bZ : 1'b0;

   always @(posedge clock4x)
     if (reset)
       begin
          state <= 0;
          ack <= 0;
       end
     else
       case (state)
         8'h00: // idle
           begin
              scl <= 1'b1;
              sdai <= 1'b1;
              ack <= 1'b0;
              idle <= 1'b1;
              if (load)
                begin
                   ldata <= data;
                   ack <= 1'b1;
                   state <= state+1;
                end
           end
         8'h01: // Start
           begin
              ack <= 1'b0;
              idle <= 1'b0;
              sdai <= 1'b0;
              state <= state+1;
           end
         8'h02:
           begin
              scl <= 1'b0;
              state <= state+1;
           end
         8'h03: // Send bit 7
           begin
              ack <= 1'b0;
              sdai <= ldata[7];
              state <= state+1;
           end
         8'h04:
           begin
              scl <= 1'b1;
              state <= state+1;
           end
         8'h05:
           begin
              state <= state+1;
           end
         8'h06:
           begin
              scl <= 1'b0;
              state <= state+1;
           end
         8'h07:
           begin
              sdai <= ldata[6];
              state <= state+1;
           end
```

```
          8'h08:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h09:
            begin
               state <= state+1;
            end
          8'h0A:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h0B:
            begin
               sdai <= ldata[5];
               state <= state+1;
            end
          8'h0C:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h0D:
            begin
               state <= state+1;
            end
          8'h0E:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h0F:
            begin
               sdai <= ldata[4];
               state <= state+1;
            end
          8'h10:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h11:
            begin
               state <= state+1;
            end
          8'h12:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h13:
            begin
               sdai <= ldata[3];
               state <= state+1;
            end
          8'h14:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h15:
            begin
               state <= state+1;
            end
          8'h16:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h17:
            begin
```

```
            sdai <= ldata[2];
            state <= state+1;
        end
      8'h18:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h19:
        begin
            state <= state+1;
        end
      8'h1A:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h1B:
        begin
            sdai <= ldata[1];
            state <= state+1;
        end
      8'h1C:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h1D:
        begin
            state <= state+1;
        end
      8'h1E:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h1F:
        begin
            sdai <= ldata[0];
            state <= state+1;
        end
      8'h20:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h21:
        begin
            state <= state+1;
        end
      8'h22:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h23: // Acknowledge bit
        begin
            state <= state+1;
        end
      8'h24:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h25:
        begin
            state <= state+1;
        end
      8'h26:
        begin
            scl <= 1'b0;
            if (load)
              begin
```

```
                    ldata <= data;
                    ack <= 1'b1;
                    state <= 3;
                  end
                else
                  state <= state+1;
            end
          8'h27:
            begin
               sdai <= 1'b0;
               state <= state+1;
            end
          8'h28:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h29:
            begin
               sdai <= 1'b1;
               state <= 0;
            end
        endcase

endmodule
```

```
module vram_display #(parameter XOFFSET = 0, YOFFSET = 0) (reset,clk,hcount,vcount,v
r_pixel,
                    vram_addr,vram_read_data);

    input reset, clk;
    input [10:0] hcount;
    input [9:0]          vcount;
    output reg [29:0] vr_pixel;
    output [18:0] vram_addr;
    input [35:0]  vram_read_data;

    wire[10:0] x;
    wire[9:0] y;
    assign x = hcount - XOFFSET;
    assign y = vcount - YOFFSET;

    //forecast hcount & vcount 2 clock cycles ahead to get data from ZBT

    // for 1024x720 resolution:
    //wire [10:0] hcount_f = (x >= 1048) ? x - 1048 : (x + 2);
    //wire [9:0] vcount_f = (x >= 1048) ? ((y == 805) ? 0 : y + 1) : y;

    // for 800x600 resolution:
    wire [10:0] hcount_f = (x >= 840) ? x - 840 : (x + 2);
    wire [9:0] vcount_f = (x >= 840) ? ((y == 627) ? 0 : y + 1) : y;

    reg [18:0] vram_addr;
    always@(*) begin
      if(hcount_f < 640 && vcount_f < 480) begin
        vram_addr = {vcount_f[8:0], hcount_f[9:0]};
        vr_pixel = vram_read_data[29:0];
      end
      else begin
        vr_pixel = {10'd0,10'd512,10'd512};
      end
    end

endmodule // vram_display
```

```
module vram_flash_img #(parameter XOFFSET = 0, YOFFSET = 0) (reset,clk,hcount,vcount
,vr_pixel,
                    vram_addr,vram_read_data);

   input reset, clk;
   input [10:0] hcount;
   input [9:0]  vcount;
   output reg [29:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

        wire[10:0] x;
        wire[9:0] y;
        assign x = hcount - XOFFSET;
        assign y = vcount - YOFFSET;
//   //forecast hcount & vcount 2 clock cycles ahead to get data from ZBT
//   wire [10:0] hcount_f = (x >= 1048) ? x - 1048 : (x + 2);
//   wire [9:0] vcount_f = (x >= 1048) ? ((y == 805) ? 0 : y + 1) : y;
//
//      reg [18:0] vram_addr;
//      always@(*) begin
//              if(hcount_f < 640 && vcount_f < 480) begin
//                      vram_addr = {vcount_f[8:0], hcount_f[9:0]};
//                      vr_pixel = vram_read_data[29:0];
//              end
//              else begin
//                      vr_pixel = {10'd0,10'd512,10'd512};
//              end
//      end
   //forecast hcount & vcount 2 clock cycles ahead to get data from ZBT
   //wire [10:0] hcount_f =(x + 2);
        //wire [10:0] hcount_f = (x >= 1048) ? x - 1048 : (x + 2);
   //wire [8:0] vcount_f =  y;
        //wire [9:0] vcount_f = (x >= 1048) ? ((y == 805) ? 0 : y + 1) : y;

  wire [10:0] hcount_f = (x >= 840) ? x - 840 : (x + 2);
  wire [9:0] vcount_f = (x >= 840) ? ((y == 627) ? 0 : y + 1) : y;

        //wire [10:0] hcount_f = (x == 1342) ? 0 : (x == 1343) ? 1 : (x+2);
   //wire [9:0] vcount_f = (x == 1343) ? ( (y == 804) ? 0 : (y == 805) ? 1 : y+1 ) :
 y;

        //reg [10:0] hcount_fold=0;
        reg [18:0] vram_addr=0;
        //wire [9:0] vcount_f;
        //reg [14:0] hcount_next=0;
        always@(posedge clk) begin
                if((hcount_f < 640) && vcount_f < 480) begin
//                      if (~(vcount_f!=vcount_fold)&&((hcount_f!=hcount_fold)&&~(hc
ount_f==0)) //hcount changed
//                              linecounter<=linecounter+1;
//                      if (hcount_f==0) linecounter<=0;
//                      if (vcount_f!=vcount_fold)&&vcount_f!=0) begin //vcount chan
ged
//                              linecounter<=0;
//                              vertcount<=vertcount+640; end
//                      if (vcount_f==0) begin
//                              vertcount<=0;end
                        //vram_addr = (vcount_f<<9)+(vcount_f<<7)+hcount_f[10:0];
                        //vram_addr = ((({vcount_f[8:0], hcount_f[9:0]})<<3)-
                                //{vcount_f[8:0], hcount_f[9:0]})>>3;
                        vram_addr <= hcount_f[9:0]+vcount_f[8:0]*640;
                        //hcount_next<=((hcount_f[9:0]+1)*5)>>2;
                        //vram_addr <= {vcount_f[8:0]>>1, hcount_next[9:0]};
                        //vram_addr <= {vcount_f[8:0]>>1, hcount_f[9:0]};
                        //if ((hcount_f==0)&&(vcount_f==0)) vram_addr<=0;
                        //else vram_addr<=vram_addr+1;
                        //vram_addr <= vertcount+linecounter;
                        vr_pixel <= vram_read_data[29:0];
                        //vram_addr <= {vcount_f[8:0]>>1, hcount_f[9:0]>>1};
                        //vountnext<=640*(vcount_f+1)
                end
                else begin
```

```
                        vr_pixel <= {10'd0,10'd0,10'd0};
                end
        end

endmodule // vram_display
```

```
//////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (800 x 600 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output       vsync;
   output       hsync;
   output       blank;

   reg    hsync,vsync,hblank,vblank,blank;
   reg [10:0]    hcount;    // pixel number on current line
   reg [9:0] vcount;     // line number

   // horizontal: 1056 pixels total
   // display 800 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 799);
   assign    hsyncon = (hcount == 839);
   assign    hsyncoff = (hcount == 967);
   assign    hreset = (hcount == 1055);

   // vertical: 628 lines total
   // display 600 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 599);
   assign    vsyncon = hreset & (vcount == 600);
   assign    vsyncoff = hreset & (vcount == 604);
   assign    vreset = hreset & (vcount == 627);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule

/*//////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output       vsync;
   output       hsync;
   output       blank;

   reg    hsync,vsync,hblank,vblank,blank;
   reg [10:0]    hcount;    // pixel number on current line
   reg [9:0] vcount;     // line number

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 1023);
   assign    hsyncon = (hcount == 1047);
   assign    hsyncoff = (hcount == 1183);
   assign    hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
```

```
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 767);
   assign    vsyncon = hreset & (vcount == 776);
   assign    vsyncoff = hreset & (vcount == 782);
   assign    vreset = hreset & (vcount == 805);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule*/
```

```
///*/*************************************************************************
// **
// ** Module: ycrcb2rgb
// **
// ** Generic Equations:
// *************************************************************************/
//
module ycrcb2rgb (Y, Cr, Cb, R, G, B, clk, rst);

output [7:0]  R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
      Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
      end
   else
      begin
          Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
      end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     A_int <= (const2 * (Cr_reg - 'd512));
     B1_int <= (const3 * (Cr_reg - 'd512));
     B2_int <= (const4 * (Cb_reg - 'd512));
     C_int <= (const5 * (Cb_reg - 'd512));
     end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     R_int <= X_int + A_int;
     G_int <= X_int - B1_int - B2_int;
     B_int <= X_int + C_int;
     end



/*always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
```

```
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     R_int <= X_int + (const2 * (Cr_reg - 'd512));
     G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
     B_int <= X_int + (const5 * (Cb_reg - 'd512));
     end

*/
/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

```
module zbt_6111_sample(beep, audio_reset_b,
                       ac97_sdata_out, ac97_sdata_in, ac97_synch,
                ac97_bit_clock,

                vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
                vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
                vga_out_vsync,

                tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
                tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
                tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

                tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
                tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
                tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
                tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

                ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
                ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

                ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
                ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

                clock_feedback_out, clock_feedback_in,

                flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                flash_reset_b, flash_sts, flash_byte_b,

                rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

                mouse_clock, mouse_data, keyboard_clock, keyboard_data,

                clock_27mhz, clock1, clock2,

                disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_in,

                button0, button1, button2, button3, button_enter, button_right,
                button_left, button_down, button_up,

                switch,

                led,

                user1, user2, user3, user4,

                daughtercard,

                systemace_data, systemace_address, systemace_ce_b,
                systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

                analyzer1_data, analyzer1_clock,
                analyzer2_data, analyzer2_clock,
                analyzer3_data, analyzer3_clock,
                analyzer4_data, analyzer4_clock);

  `include "param.v"

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
          vga_out_hsync, vga_out_vsync;

   output [9:0] tv_out_ycrcb;
   output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
          tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
          tv_out_subcar_reset;

   input  [19:0] tv_in_ycrcb;
   input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
          tv_in_hff, tv_in_aff;
```

```
   output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
          tv_in_reset_b, tv_in_clock;
   inout  tv_in_i2c_data;

   inout  [35:0] ram0_data;
   output [18:0] ram0_address;
   output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
   output [3:0] ram0_bwe_b;

   inout  [35:0] ram1_data;
   output [18:0] ram1_address;
   output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
   output [3:0] ram1_bwe_b;

   input  clock_feedback_in;
   output clock_feedback_out;

   inout  [15:0] flash_data;
   output [23:0] flash_address;
   output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
   input  flash_sts;

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

   output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
   input  disp_data_in;
   output  disp_data_out;

   input  button0, button1, button2, button3, button_enter, button_right,
          button_left, button_down, button_up;
   input  [7:0] switch;
   output [7:0] led;

   inout [31:0] user1, user2, user3, user4;

   inout [43:0] daughtercard;

   inout  [15:0] systemace_data;
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
                 analyzer4_data;
   output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

   ////////////////////////////////////////////////////////////////////////
   //
   // I/O Assignments
   //
   ////////////////////////////////////////////////////////////////////////

   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   assign ac97_sdata_out = 1'b0;
/*
*/
   // ac97_sdata_in is an input

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
```

```
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   //assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   //assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = clock_27mhz;//1'b0;
   //assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs

/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;    // clock enable
*/

/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

/**********/

   //assign ram1_data = 36'hZ;
   //assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   //assign ram1_clk = 1'b0;

   //These values has to be set to 0 like ram0 if ram1 is used.
   //assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b0;
   assign ram1_oe_b = 1'b0;
   //assign ram1_we_b = 1'b0;
   assign ram1_bwe_b = 4'h0;

   // clock_feedback_out will be assigned by ramclock
   // assign clock_feedback_out = 1'b0;  //2011-Nov-10
   // clock_feedback_in is an input

   // Flash ROM
   //assign flash_data = 16'hZ;
   //assign flash_address = 24'h0;
   //assign flash_ce_b = 1'b1;
   //assign flash_oe_b = 1'b1;
   //assign flash_we_b = 1'b1;
   //assign flash_reset_b = 1'b0;
   //assign flash_byte_b = 1'b1;
   // flash_sts is an input

   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
   // rs232_rxd and rs232_cts are inputs

   // PS/2 Ports
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

   // LED Displays
/*
```

```
   assign disp_blank = 1'b1;
   assign disp_clock = 1'b0;
   assign disp_rs = 1'b0;
   assign disp_ce_b = 1'b1;
   assign disp_reset_b = 1'b0;
   assign disp_data_out = 1'b0;
*/
   // disp_data_in is an input

   // Buttons, Switches, and Individual LEDs
   //lab3 assign led = 8'hFF;
   // button0, button1, button2, button3, button_enter, button_right,
   // button_left, button_down, button_up, and switches are inputs

   // User I/Os
   assign user1[31:1] = 31'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

/*   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));*/

//   wire clk = clock_65mhz;  // gph 2011-Nov-10

   ////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory
   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 40.5MHz)
   wire clock_40mhz_unbuf,clock_40mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));
   //wire clk = clock_40mhz;

        wire locked;
        //assign clock_feedback_out = 0; // gph 2011-Nov-10

  wire clock_65mhz = clock_40mhz; // when switching from 1024x768 to 800x600
```

```
   ramclock rc(.ref_clock(clock_65mhz), .fpga_clock(clk),
                                        .ram0_clock(ram0_clk),
                                        .ram1_clock(ram1_clk),   //uncomment if ram1
 is used
                                        .clock_feedback_in(clock_feedback_in),
                                        .clock_feedback_out(clock_feedback_out), .lo
cked(locked));


   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
                   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // ENTER button is user reset ---> (DISABLED)
   wire reset,user_reset;
   //debounce db1(power_on_reset, clk, ~button_enter, user_reset);
        assign user_reset = 1'b0;
   assign reset = user_reset | power_on_reset;

   // display module for debugging
   reg [63:0] dispdata;
   display_16hex hexdisp1(reset, clk, dispdata,
                          disp_blank, disp_clock, disp_rs, disp_ce_b,
                          disp_reset_b, disp_data_out);

   // generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   xvga xvga1(clk,hcount,vcount,hsync,vsync,blank);

   // wire up to ZBT ram
   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
   wire [18:0] vram_addr;
   wire vram_we;

   wire ram0_clk_not_used;
   zbt_6111 zbt0(clk, 1'b1, vram_we, vram_addr,
                 vram_write_data, vram_read_data,
                 ram0_clk_not_used,   //to get good timing, don't connect ram_clk
to zbt_6111
                 ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

   // generate pixel value from reading ZBT memory
   wire [29:0]  vr_pixel;
   wire [18:0]  display_addr;

   vram_display #(1,0) /*#(192,144)*/ vd1(reset,clk,hcount,vcount,vr_pixel,
                  display_addr,vram_read_data);

   // ADV7185 NTSC decoder interface code
   // adv7185 initialization module
   adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
                       .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                       .tv_in_i2c_clock(tv_in_i2c_clock),
                       .tv_in_i2c_data(tv_in_i2c_data));

   wire [29:0] ycrcb;   // video data (luminance, chrominance)
   wire [2:0] fvh;      // sync for field, vertical, horizontal
   wire       dv;       // data valid

   ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                       .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                       .ycrcb(ycrcb), .f(fvh[2]),
                       .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

   // code to write NTSC data to video memory
   wire [18:0] ntsc_addr;
   wire [35:0] ntsc_data;
```

```
   wire ntsc_we;
   ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, ycrcb[29:0],
                    ntsc_addr, ntsc_data, ntsc_we);

  ////////////////////////////////////////////////////////////////////////
  //
  // Input Buttons & Switches
  //
  ////////////////////////////////////////////////////////////////////////

  wire sw_ntsc, enhance_en, filters_en, store_bram;   // editing & storage
  wire text_en, graphics_en, move_sw, custom_text_en; // text & graphics
  wire up, down, left, right, enter;
  wire select0, select1, select2, select3;

  debounce db2(.reset(reset),.clk(clock_65mhz),.noisy(~switch[7]),.clean(sw_ntsc));
  debounce db3(.reset(reset),.clk(clock_65mhz),.noisy(switch[6]),.clean(enhance_en))
;
  debounce db4(.reset(reset),.clk(clock_65mhz),.noisy(switch[5]),.clean(filters_en))
;
  debounce db5(.reset(reset),.clk(clock_65mhz),.noisy(switch[4]),.clean(text_en));
  debounce db6(.reset(reset),.clk(clock_65mhz),.noisy(switch[3]),.clean(graphics_en)
);
  debounce db7(.reset(reset),.clk(clock_65mhz),.noisy(switch[2]),.clean(move_sw));
  debounce db8(.reset(reset),.clk(clock_65mhz),.noisy(switch[1]),.clean(custom_text_
en));
  debounce db9(.reset(reset),.clk(clock_65mhz),.noisy(switch[0]),.clean(store_bram))
;

  debounce db10(.reset(reset),.clk(clock_65mhz),.noisy(~button_up),.clean(up));
  debounce db11(.reset(reset),.clk(clock_65mhz),.noisy(~button_down),.clean(down));
  debounce db12(.reset(reset),.clk(clock_65mhz),.noisy(~button_left),.clean(left));
  debounce db13(.reset(reset),.clk(clock_65mhz),.noisy(~button_right),.clean(right))
;
  debounce db14(.reset(reset),.clk(clock_65mhz),.noisy(~button_enter),.clean(enter))
;

  debounce db15(.reset(reset),.clk(clock_65mhz),.noisy(~button0),.clean(select0));
  debounce db16(.reset(reset),.clk(clock_65mhz),.noisy(~button1),.clean(select1));
  debounce db17(.reset(reset),.clk(clock_65mhz),.noisy(~button2),.clean(select2));
  debounce db18(.reset(reset),.clk(clock_65mhz),.noisy(~button3),.clean(select3));

  wire move_text_en;      // when move_sw is low
  wire move_graphics_en;  // when move_sw is high
  assign {move_text_en, move_graphics_en} = {~move_sw, move_sw};

  wire sw_ntsc_n = ~sw_ntsc;

  wire sel_all = select0 && select1 && select2 && select3;
  wire but_all = up && down && left && right && enter;
  wire sw_all = switch[7] && switch[6] && switch[5] && switch[4] && switch[3] && swi
tch[2] && switch[1] && switch[0];

  wire sel_on = select0 || select1 || select2 || select3;

  ////////////////////////////////////////////////////////////////////////

   wire [35:0]  write_data = ntsc_data;
        //address is either chosen by camera or display
   assign       vram_addr = sw_ntsc ? ntsc_addr : display_addr;
        //write enable when in write mode and camera wants to write
   assign       vram_we = sw_ntsc & ntsc_we;
   assign       vram_write_data = write_data;

  ////////////////////////////////////////////////////////////////////////
  //
  // Main FSM
  //
  ////////////////////////////////////////////////////////////////////////

  wire [2:0] fsm_state;
  wire sw_ntsc_falling;
```

```
  main_fsm fsm1(
    .clk              (clk),
    .rst              (reset),
    .sw_ntsc          (sw_ntsc_n),
    .enter            (enter),
    .store_bram       (store_bram),
    .fsm_state        (fsm_state),
    .sw_ntsc_falling  (sw_ntsc_falling)
  );

  //////////////////////////////////////////////////////////////////////////////
  //
  // Background Selection
  //
  //////////////////////////////////////////////////////////////////////////////

  // determine which background was selected
  reg [2:0] background_q;
  wire [2:0] background = background_q;

  always @(posedge clk) begin
    if ((fsm_state == SEL_BKGD) && select0) background_q <= PARIS;
    if ((fsm_state == SEL_BKGD) && select1) background_q <= ROME;
    if ((fsm_state == SEL_BKGD) && select2) background_q <= AMAZON;
    if ((fsm_state == SEL_BKGD) && select3) background_q <= LONDON;
    if ((fsm_state == SEL_BKGD) && sel_all) background_q <= NO_BKD;
  end

  //////////////////////////////////////////////////////////////////////////////
  //
  // Flash & Background Storage
  //
  //////////////////////////////////////////////////////////////////////////////

        //ALL THE FLASH STUFF
        wire [15:0] wdata; //write data. not needed here
        wire writemode;
        wire dowrite;
        wire [22:0] raddr; // address of where we want to read from flash (playing f
rom flash)
        wire [15:0] frdata;     //data being read
        wire doread; // tell flash to read from memory
        wire busy; // flash is busy, don't read/write when asserted
        wire [11:0] fsmstate; //for debugging
        wire dots;
        wire flashreset; //done want this asserted easily/at all
        assign flashreset=sw_all && but_all && sel_all;
        reg [2:0] pictsel=4;
  always @(posedge clock_27mhz) pictsel<={1'b0, background[1:0]};
        wire start;
        assign start = (fsm_state == SEL_BKGD) && sel_on;
        wire [22:0] flashaddr;
        wire fifo_rd_en, fifo_wr_en, fifo_empty, fifo_full;
        wire [18:0] display_addr_flash_img;
        wire [35:0] vram_read_data_flash_img;
        wire [15:0] fifodata;
        wire [18:0] zbtaddr;
        reg [18:0]  zbtwrite=0;
        wire loaded;
        wire zbt_we;
        reg busy_old;
        reg readytostore=0;
        assign zbt_we= readytostore;
        wire image_loaded = (zbtwrite == 19'd307200);
  assign zbtaddr= loaded ? display_addr_flash_img : zbtwrite; //reading vs writing a
dresses
        wire ram1_clk_not_used;
        reg [15:0] datatostore=0; //16 bit color pixels
        zbt_6111 zbt_6111_flash_img(clk, 1'b1,zbt_we,zbtaddr, {20'd0,datatostore}, v
ram_read_data_flash_img,
                                    ram1_clk_not_used, ram1_we_b, ram1_address, ram1_data, r
am1_cen_b);
```

```
        flash_manager flash(.clock(clock_27mhz), .reset(flashreset), .dots(dots), .w
ritemode(writemode), .wdata(wdata),
                                        .dowrite(dowrite), .raddr(raddr), .frdata(
frdata), .doread(doread), .busy(busy),
                                        .flash_data(flash_data), .flash_address(fl
ash_address), .flash_ce_b(flash_ce_b),
                                        .flash_oe_b(flash_oe_b), .flash_we_b(flash
_we_b), .flash_reset_b(flash_reset_b),
                                        .flash_sts(flash_sts), .flash_byte_b(flash
_byte_b), .fsmstate(fsmstate));


        flashreader flashreader(clock_27mhz,flashreset,busy,start,pictsel,raddr,
                                                                        writemode,do
write,doread,wdata, loaded);


        reg [15:0] frdata_q=0;
        reg [18:0] frdata_ctr_q = 0;
        reg [18:0] datathing=0;

  always @(posedge clk) begin
                busy_old<=busy;
                if (start) zbtwrite<=0;
                if (~start && busy_old && ~busy && ~loaded) begin //edge detection o
n busy signal
                                datatostore<=frdata;
                                readytostore<=1;
                                zbtwrite<=zbtwrite+1;
                                datathing<=datathing+1;
                        end
                else readytostore<=0;
        end

  wire [29:0]   bkgd_vr_pixel;

        vram_flash_img #(0,0) vd2(reset,clk,hcount,vcount,bkgd_vr_pixel,
                    display_addr_flash_img,vram_read_data_flash_img);

  ////////////////////////////////////////////////////////////////////////////
  //
  // Custom Text
  //
  ////////////////////////////////////////////////////////////////////////////

  wire [CUSTOM_TEXT_MAXLEN*8-1:0] char_array; // user-entered characters
  wire char_array_rdy;  // after user presses enter when entering keyboard input
  wire [5:0] num_char; // num char user has entered via keyboard

  custom_text #(CUSTOM_TEXT_MAXLEN) custom_text1(
    .clock_27mhz          (clock_27mhz),
    .reset                (reset),
    .custom_text_en       (custom_text_en),
    .fsm_state            (fsm_state),
    .button_enter         (enter),
    .keyboard_clock       (keyboard_clock),
    .keyboard_data        (keyboard_data),
    .char_array           (char_array),
    .char_array_rdy       (char_array_rdy),
    .num_char             (num_char)
  );

  ////////////////////////////////////////////////////////////////////////////
  //
  // Output Pixel
  //
  ////////////////////////////////////////////////////////////////////////////

  // VGA signals
  wire [23:0] pixel_out;
  wire blank_out, hsync_out, vsync_out;

  // BRAM signals
```

```
  wire [10:0] h_offset;
  wire in_display_bram;
  wire [7:0]  bram_dout;
  wire [1:0]  bram_state;

  // Text & Graphics Coordinates
  wire [10:0] text_x_pos, graphics_x_pos;
  wire [9:0]  text_y_pos, graphics_y_pos;

  // Chroma-Key Compositing
  wire [7:0] thr_range, h_thr, s_thr, v_thr;

  // Image Enhancement
  wire [7:0] s_offset, v_offset;
  wire s_dir, v_dir;

  // User Selections
  wire [2:0] selected_filter;
  wire [1:0] selected_graphic;

  // Background Pixel
  wire [23:0] vr_bkgd;
  assign vr_bkgd[23:16] = {bkgd_vr_pixel[15:11], 3'd0};
  assign vr_bkgd[15:8] = {bkgd_vr_pixel[10:5], 2'd0};
  assign vr_bkgd[7:0] = {bkgd_vr_pixel[4:0], 3'd0};

  pixel_sel #(CUSTOM_TEXT_MAXLEN) pixel_sel1(
    .clk                    (clk),
    .reset                  (reset),
    // states
    .fsm_state              (fsm_state),
    .bram_state             (bram_state),
    // user switches
    .sw_ntsc                (sw_ntsc),
    .store_bram             (store_bram),
    .enhance_en             (enhance_en),
    .filters_en             (filters_en),
    .text_en                (text_en),
    .graphics_en            (graphics_en),
    .move_text_en           (move_text_en),
    .move_graphics_en       (move_graphics_en),
    .custom_text_en         (custom_text_en),
    // user buttons
    .up                     (up),
    .down                   (down),
    .left                   (left),
    .right                  (right),
    .select0                (select0),
    .select1                (select1),
    .select2                (select2),
    .select3                (select3),
    .background             (background),
    // custom text gen
    .num_char               (num_char),
    .char_array_rdy         (char_array_rdy),
    .char_array             (char_array),
    // pixel values
    .vr_pixel               (vr_pixel),
    .bram_dout              (bram_dout),
    .vr_bkgd_color          (vr_bkgd),
    // VGA timing
    .hcount                 (hcount),
    .vcount                 (vcount),
    .blank                  (blank),
    .hsync                  (hsync),
    .vsync                  (vsync),
    .h_offset               (h_offset),
    .in_display_bram        (in_display_bram),
    // VGA outputs
    .pixel_out              (pixel_out),
    .blank_out              (blank_out),
    .hsync_out              (hsync_out),
    .vsync_out              (vsync_out),
```

```
    .text_x_pos             (text_x_pos),
    .text_y_pos             (text_y_pos),
    .graphics_x_pos         (graphics_x_pos),
    .graphics_y_pos         (graphics_y_pos),
    // hex display
    .thr_range              (thr_range),
    .h_thr                  (h_thr),
    .v_thr                  (v_thr),
    .s_thr                  (s_thr),
    // image enhancement
    .s_offset               (s_offset),
    .v_offset               (v_offset),
    .s_dir                  (s_dir),
    .v_dir                  (v_dir),
    // user selections
    .selected_filter        (selected_filter),
    .selected_graphic       (selected_graphic)
  );

  ////////////////////////////////////////////////////////////////////////
  //
  // Display Signals for BRAM Frame
  //
  ////////////////////////////////////////////////////////////////////////

  // offset hcount to sync first pixel being stored in BRAM with the first
  // image pixel coming out on the VGA pixel signal after image processing

  assign h_offset = (!filters_en) ? SYNC_DLY :
                    (selected_filter == SEPIA) ? SYNC_DLY_SEP :
                    (selected_filter == INVERT) ? SYNC_DLY_INV :
                    (selected_filter == GRAYSCALE) ? SYNC_DLY_GRY :
                    (selected_filter == EDGE) ? SYNC_DLY+10'd10 :
                    (selected_filter == CARTOON) ? SYNC_DLY+10'd10 : SYNC_DLY;

  wire hcount_in_display_bram = (hcount >= h_offset) && (hcount < (H_MAX_DISPLAY+h_o
ffset));
  wire vcount_in_display_bram = (vcount >= V_OFFSET) && (vcount < (V_MAX_DISPLAY+V_O
FFSET));

  // create a signal that will determine which pixels to write to BRAM + when to rea
d them out
  assign in_display_bram = hcount_in_display_bram && vcount_in_display_bram;

  ////////////////////////////////////////////////////////////////////////
  //
  // BRAM-Based Frame Buffer
  //
  ////////////////////////////////////////////////////////////////////////

  wire [17:0] bram_tx_counter; // serves as read address when transmitting stored da
ta to PC

  frame_bram_ifc frame_bram(
    .clk          (clk),
    .rst          (1'b0),
    .store_bram   (store_bram),
    .hcount       (hcount),
    .vcount       (vcount),
    .hoffset      (h_offset),
    .voffset      (V_OFFSET),
    .in_display   (in_display_bram),
    .pixel_out    (pixel_out),
    .fsm_state    (fsm_state),
    .tx_counter   (bram_tx_counter),
    .bram_dout    (bram_dout),
    .bram_state   (bram_state)
  );

  ////////////////////////////////////////////////////////////////////////
  //
  // UART TX
  //
```

```
  ////////////////////////////////////////////////////////////////////////

  // R/G/B byte transmission states
  localparam TX_RED = 2;   // if rgb_tx_state_q == TX_RED, currently transmitting R
byte
  localparam TX_GREEN = 0; // if rgb_tx_state_q == TX_GREEN, currently transmitting
G byte
  localparam TX_BLUE = 1;  // if rgb_tx_state_q == TX_BLUE, currently transmitting B
 byte

  reg [1:0] rgb_tx_state_q = 0; // used in transmitting separate R/G/B bytes over UA
RT

  reg [17:0] uart_tx_counter_q = 0;
  assign bram_tx_counter = (uart_tx_counter_q < 18'd256000) ? uart_tx_counter_q : 0;

  // UART signals
  wire uart_reset, uart_tx_en;
  wire [7:0] uart_tx_data_in;
  wire uart_tx_bit_out;
  wire [3:0] uart_tx_bit_ctr;
  wire uart_tx_busy;

  uart_tx uart_tx1(
    .clk      (clock_65mhz),
    .rst      (uart_reset),
    .tx_en    (uart_tx_en),
    .data_in  (uart_tx_data_in),
    .bit_out  (uart_tx_bit_out),
    .tx_busy  (uart_tx_busy),
    .bit_ctr  (uart_tx_bit_ctr)
   );

  assign uart_reset = (fsm_state == FSM_IDLE);

  wire frame_tx_done = (uart_tx_counter_q == 18'd256000); // transmitted  full frame
  assign uart_tx_en = (fsm_state == SEND_TO_PC) && !frame_tx_done;

  assign uart_tx_data_in = (rgb_tx_state_q == TX_RED) ? {bram_dout[7:5], 5'd0} :
                           (rgb_tx_state_q == TX_GREEN) ? {bram_dout[4:2], 5'd0} :
                           (rgb_tx_state_q == TX_BLUE) ? {bram_dout[1:0], 6'd0} : 8
'd0;

  always @(posedge clk) begin
    if (uart_reset) begin
      uart_tx_counter_q <= 0;
      rgb_tx_state_q <= 0;
    end else if (uart_tx_bit_ctr == 4'd1) begin
      // when UART module is almost done transmitting the current byte and not all
      // of the image/frame has been transmitted, determine next R/G/B byte
      if (uart_tx_en) rgb_tx_state_q <= (rgb_tx_state_q == 2'b10) ? 2'b00 : rgb_tx_s
tate_q+1;
      // if all three bytes for a pixel have been sent, get next pixel in image/fram
e
      if (rgb_tx_state_q == 2'b10) uart_tx_counter_q <= uart_tx_counter_q + 1'b1;
    end
  end

  assign user1[0] = uart_tx_bit_out; // send the data bit to wire

  ////////////////////////////////////////////////////////////////////////////
  //
  // VGA Output
  //
  ////////////////////////////////////////////////////////////////////////////

  wire h_in_start_disp = (hcount > 20) && (hcount < 620);
  wire v_in_start_disp = (vcount > 20) && (vcount < 300);
  wire in_start_disp = h_in_start_disp && v_in_start_disp;

   // In order to meet the setup and hold times of the AD7125, we send it ~clk.
//   assign vga_out_red = (fsm_state == FSM_IDLE) ? ((in_start_disp) ? vr_bkgd[23:16
] : 0) : pixel_out[23:16];
```

```
//   assign vga_out_green = (fsm_state == FSM_IDLE) ? ((in_start_disp) ? vr_bkgd[15:
8] : 0) : pixel_out[15:8];
//   assign vga_out_blue = (fsm_state == FSM_IDLE) ? ((in_start_disp) ? vr_bkgd[7:0]
 : 0) : pixel_out[7:0];
   assign vga_out_red = pixel_out[23:16];
   assign vga_out_green = pixel_out[15:8];
   assign vga_out_blue = pixel_out[7:0];
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_pixel_clock = ~clk;
   assign vga_out_blank_b = ~blank_out;
   assign vga_out_hsync = hsync_out;
   assign vga_out_vsync = vsync_out;

  ////////////////////////////////////////////////////////////////////////
  //
  // LED and HEX16 Display
  //
  ////////////////////////////////////////////////////////////////////////

   assign led = ~{bram_state, background[1:0], selected_filter[1:0], selected_graphi
c};

        // Hex Display
   always @(posedge clk) begin
     case (fsm_state)
      FSM_IDLE       : begin
                         dispdata[63:60] <= 4'hF; // F = Idle / Start Screen
                         dispdata[59:0] <= 0;
                       end
      SEL_BKGD       : begin
                         dispdata[63:60] <= 4'hD; // D = Choose Background
                         // output threshold range + hue + saturation + value/brightn
ess thresholds
                         dispdata[59:0] <= {4'h0, thr_range, 4'h0, h_thr, 4'h0, s_thr
, 4'h0, v_thr};
                       end
      COLOR_EDITS    : begin
                         dispdata[63:60] <= 4'hC; // C = Change Color or Filter Effec
t
                         dispdata[59:24] <= 0;
                         dispdata[23:0] <= {3'b000, s_dir, s_offset, 3'b000, v_dir, v
_offset};
                       end
      ADD_EDITS      : begin
                         dispdata[63:60] <= 4'hA; // A = Add Text or Graphics
                         // output number of characters in custom text input
                         dispdata[59:48] <= {8'h00, 3'b000, num_char};
                         // output coordinates of overlapping text
                         dispdata[48:24] <= {1'b0, text_x_pos, 2'b00, text_y_pos};
                         // output coordinates of overlapping graphics
                         dispdata[23:0] <= {1'b0, graphics_x_pos, 2'b00, graphics_y_p
os};
                       end
      SAVE_TO_BRAM   : begin
                         dispdata[63:60] <= 4'hB; // B = Store in BRAM
                         dispdata[59:0] <= 0;
                       end
      SEND_TO_PC     : begin
                         dispdata[63:60] <= 4'hE; // E = Send to PC
                         dispdata[59:24] <= 0;
                         dispdata[23:0] <= {4'h0, 2'b00, uart_tx_counter_q};   // num
 of pixels transmitted
                       end
     endcase

   end

endmodule
```

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

////////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;                   // system clock
   input cen;                   // clock enable for gating ZBT cycles
   input we;                    // write enable (active HIGH)
   input [18:0] addr;           // memory address
   input [35:0] write_data;     // data to write
   output [35:0] read_data;     // data read from memory
   output        ram_clk;       // physical line to ram clock
   output        ram_we_b;      // physical line to ram we_b
   output [18:0] ram_address;   // physical line to ram address
   inout [35:0]  ram_data;      // physical line to ram data
   output        ram_cen_b;     // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
   wire          ram_cen_b = ~cen;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0]   we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;

   // create two-stage pipeline for write data

   reg [35:0]  write_data_old1;
   reg [35:0]  write_data_old2;
   always @(posedge clk)
     if (cen)
       {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

   // wire to ZBT RAM signals

   assign      ram_we_b = ~we;
   assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                                // set to zero as place holder


//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                                   // times if its clk edges equal FPGA's
                                   // so we clock it on the falling edges
                                   // and thus let data stabilize longer
   assign      ram_address = addr;
   assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
   assign      read_data = ram_data;

endmodule // zbt_6111
```

```
##############################################################################
#
# 6.111 FPGA Labkit -- Constraints File
#
# For Labkit Revision 004
#
#
# Created: Oct 30, 2004 from revision 003 contraints file
# Author: Nathan Ickes and Isaac Cambron
#
##############################################################################
#
# CHANGES FOR BOARD REVISION 004
#
# 1) Added signals for logic analyzer pods 2-4.
# 2) Expanded tv_in_ycrcy bus to 20 bits.
# 3) Renamed tv_out_sclk to tv_out_i2c_clock for consistency
# 4) Renamed tv_out_data to tv_out_i2c_data for consistency
# 5) Reversed disp_data_in and disp_data_out signals, so that "out" is an
#    output of the FPGA, and "in" is an input.
#
# CHANGES FOR BOARD REVISION 003
#
# 1) Combined flash chip enables into a single signal, flash_ce_b.
# 2) Moved SRAM feedback clock loop to FPGA pins AL28 (out) and AJ16 (in).
# 3) Moved rs232_rts to FPGA pin R3.
# 4) Moved flash_address<1> to AE14.
#
# CHANGES FOR BOARD REVISION 002
#
# 1) Moved ZBT_BANK1_CLK signal to pin Y9.
# 2) Moved user1<30> to J14.
# 3) Moved user3<29> to J13.
# 4) Added SRAM clock feedback loop between D15 and H15.
# 5) Renamed ram#_parity and ram#_we#_b signals.
# 6) Removed the constraint on "systemace_clock", since this net no longer
#    exists. The SystemACE clock is now hardwired to the 27MHz oscillator
#    on the PCB.
#
##############################################################################
#
# Complete change history (including bug fixes)
#
# 2007-Aug-16: Fixed revision history. (flash_address<1> was actually changed
#              to AE14 for revision 003.)
#
# 2005-Sep-09: Added missing IOSTANDARD attribute to "disp_data_out".
#
# 2005-Jan-23: Added a pullup to FLASH_STS
#
# 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
#              actually populated on the boards. (The boards support up to
#              256Mb devices, with 25 address lines.)
#
# 2005-Jan-23: Change history started.
#
##############################################################################

#
# Audio CODEC
#

NET "beep"           LOC="ac19" | IOSTANDARD=LVDCI_33;
NET "audio_reset_b"  LOC="ae18" | IOSTANDARD=LVTTL;
NET "ac97_sdata_out" LOC="ac18" | IOSTANDARD=LVDCI_33;
NET "ac97_sdata_in"  LOC="aj24";
NET "ac97_synch"     LOC="ac17" | IOSTANDARD=LVDCI_33;
NET "ac97_bit_clock" LOC="ah24";

#
# VGA Output
#
```

```
NET "vga_out_red<7>" LOC="ae9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<6>" LOC="ae8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<5>" LOC="ad12" | IOSTANDARD=LVTTL;
NET "vga_out_red<4>" LOC="af8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<3>" LOC="af9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<2>" LOC="ag9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<1>" LOC="ag10" | IOSTANDARD=LVTTL;
NET "vga_out_red<0>" LOC="af11" | IOSTANDARD=LVTTL;

NET "vga_out_green<7>" LOC="ah8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<6>" LOC="ah7"  | IOSTANDARD=LVTTL;
NET "vga_out_green<5>" LOC="aj6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<4>" LOC="ah6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<3>" LOC="ad15" | IOSTANDARD=LVTTL;
NET "vga_out_green<2>" LOC="ac14" | IOSTANDARD=LVTTL;
NET "vga_out_green<1>" LOC="ag8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<0>" LOC="ac12" | IOSTANDARD=LVTTL;

NET "vga_out_blue<7>" LOC="ag15" | IOSTANDARD=LVTTL;
NET "vga_out_blue<6>" LOC="ag14" | IOSTANDARD=LVTTL;
NET "vga_out_blue<5>" LOC="ag13" | IOSTANDARD=LVTTL;
NET "vga_out_blue<4>" LOC="ag12" | IOSTANDARD=LVTTL;
NET "vga_out_blue<3>" LOC="aj11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<2>" LOC="ah11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<1>" LOC="aj10" | IOSTANDARD=LVTTL;
NET "vga_out_blue<0>" LOC="ah9"  | IOSTANDARD=LVTTL;

NET "vga_out_sync_b"       LOC="aj9"  | IOSTANDARD=LVTTL;
NET "vga_out_blank_b"      LOC="aj8"  | IOSTANDARD=LVTTL;
NET "vga_out_pixel_clock" LOC="ac10" | IOSTANDARD=LVTTL;
NET "vga_out_hsync"        LOC="ac13" | IOSTANDARD=LVTTL;
NET "vga_out_vsync"        LOC="ac11" | IOSTANDARD=LVTTL;

#
# Video Output
#

NET "tv_out_ycrcb<9>" LOC="p27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<8>" LOC="r27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<7>" LOC="t29" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<6>" LOC="h26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<5>" LOC="j26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<4>" LOC="l26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<3>" LOC="m26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<2>" LOC="n26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<1>" LOC="p26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<0>" LOC="r26" | IOSTANDARD=LVDCI_33;

NET "tv_out_reset_b" LOC="g27" | IOSTANDARD=LVDCI_33;
NET "tv_out_clock" LOC="l27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_clock" LOC="j27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_data" LOC="h27" | IOSTANDARD=LVDCI_33;
NET "tv_out_pal_ntsc" LOC="j25" | IOSTANDARD=LVDCI_33;
NET "tv_out_hsync_b" LOC="n27" | IOSTANDARD=LVDCI_33;
NET "tv_out_vsync_b" LOC="m27" | IOSTANDARD=LVDCI_33;
NET "tv_out_blank_b" LOC="h25" | IOSTANDARD=LVDCI_33;
NET "tv_out_subcar_reset" LOC="k27" | IOSTANDARD=LVDCI_33;

#
# Video Input
#

NET "tv_in_ycrcb<19>" LOC="ag17";
NET "tv_in_ycrcb<18>" LOC="ag18";
NET "tv_in_ycrcb<17>" LOC="ag19";
NET "tv_in_ycrcb<16>" LOC="ag20";
NET "tv_in_ycrcb<15>" LOC="ae20";
NET "tv_in_ycrcb<14>" LOC="af21";
NET "tv_in_ycrcb<13>" LOC="ad20";
NET "tv_in_ycrcb<12>" LOC="ag23";
NET "tv_in_ycrcb<11>" LOC="aj26";
NET "tv_in_ycrcb<10>" LOC="ah26";
NET "tv_in_ycrcb<9>" LOC="w23";
```

```
NET "tv_in_ycrcb<8>" LOC="v23";
NET "tv_in_ycrcb<7>" LOC="u23";
NET "tv_in_ycrcb<6>" LOC="t23";
NET "tv_in_ycrcb<5>" LOC="t26";
NET "tv_in_ycrcb<4>" LOC="t24";
NET "tv_in_ycrcb<3>" LOC="r25";
NET "tv_in_ycrcb<2>" LOC="l30";
NET "tv_in_ycrcb<1>" LOC="m31";
NET "tv_in_ycrcb<0>" LOC="m30";

NET "tv_in_data_valid" LOC="ah25";
NET "tv_in_line_clock1" LOC="ad16" | IOSTANDARD=LVDCI_33;
NET "tv_in_line_clock2" LOC="ad17" | IOSTANDARD=LVDCI_33;
NET "tv_in_aef" LOC="aj23";
NET "tv_in_hff" LOC="ah23";
NET "tv_in_aff" LOC="aj22";
NET "tv_in_i2c_clock" LOC="ad21" | IOSTANDARD=LVDCI_33;
NET "tv_in_i2c_data" LOC="ad19" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_read" LOC="ac22" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_clock" LOC="ag22" | IOSTANDARD=LVDCI_33;
NET "tv_in_iso" LOC="aj27" | IOSTANDARD=LVDCI_33;
NET "tv_in_reset_b" LOC="ag25" | IOSTANDARD=LVDCI_33;
NET "tv_in_clock" LOC="ab21" | IOSTANDARD=LVDCI_33;

#
# SRAMs
#

NET "ram0_data<35>" LOC="ab25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<34>" LOC="ah29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<33>" LOC="ag28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<32>" LOC="ag29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<31>" LOC="af27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<30>" LOC="af29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<29>" LOC="af28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<28>" LOC="ae28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<27>" LOC="ad25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<26>" LOC="aa25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<25>" LOC="ah30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<24>" LOC="ah31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<23>" LOC="ag30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<22>" LOC="ag31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<21>" LOC="af30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<20>" LOC="af31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<19>" LOC="ae30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<18>" LOC="ae31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<17>" LOC="y27"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<16>" LOC="aa28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<15>" LOC="y29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<14>" LOC="y28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<13>" LOC="w29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<12>" LOC="w28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<11>" LOC="v28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<10>" LOC="u29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<9>"  LOC="u28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<8>"  LOC="aa27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<7>"  LOC="ad31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<6>"  LOC="ac30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<5>"  LOC="ac31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<4>"  LOC="ab30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<3>"  LOC="ab31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<2>"  LOC="aa30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<1>"  LOC="aa31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<0>"  LOC="y30"  | IOSTANDARD=LVDCI_33 | NODELAY;

NET "ram0_address<18>" LOC="v31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<17>" LOC="w31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<16>" LOC="ad28" | IOSTANDARD=LVDCI_33;
NET "ram0_address<15>" LOC="ad29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<14>" LOC="ac24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<13>" LOC="ad26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<12>" LOC="ad27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<11>" LOC="ac27" | IOSTANDARD=LVDCI_33;
```

```
NET "ram0_address<10>" LOC="ab27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<9>" LOC="y31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<8>" LOC="w30" | IOSTANDARD=LVDCI_33;
NET "ram0_address<7>" LOC="y26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<6>" LOC="y25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<5>" LOC="ab24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<4>" LOC="ac25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<3>" LOC="aa26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<2>" LOC="aa24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<1>" LOC="ab29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<0>" LOC="ac26" | IOSTANDARD=LVDCI_33;

NET "ram0_adv_ld" LOC="v26" | IOSTANDARD=LVDCI_33;
NET "ram0_clk" LOC="u30" | IOSTANDARD=LVDCI_33;
NET "ram0_cen_b" LOC="u25" | IOSTANDARD=LVDCI_33;
NET "ram0_ce_b" LOC="w26" | IOSTANDARD=LVDCI_33;
NET "ram0_oe_b" LOC="v25" | IOSTANDARD=LVDCI_33;
NET "ram0_we_b" LOC="u31" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<0>" LOC="v27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<1>" LOC="u27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<2>" LOC="w27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<3>" LOC="u26" | IOSTANDARD=LVDCI_33;

NET "ram1_data<35>" LOC="aa9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<34>" LOC="ah2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<33>" LOC="ah1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<32>" LOC="ag2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<31>" LOC="ag1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<30>" LOC="af2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<29>" LOC="af1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<28>" LOC="ae2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<27>" LOC="ae1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<26>" LOC="ab9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<25>" LOC="ah3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<24>" LOC="ag4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<23>" LOC="ag3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<22>" LOC="af4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<21>" LOC="af3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<20>" LOC="ae4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<19>" LOC="ae5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<18>" LOC="ad5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<17>" LOC="v2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<16>" LOC="ad1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<15>" LOC="ac2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<14>" LOC="ac1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<13>" LOC="ab2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<12>" LOC="ab1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<11>" LOC="aa2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<10>" LOC="aa1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<9>" LOC="y2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<8>" LOC="v4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<7>" LOC="ac3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<6>" LOC="ac4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<5>" LOC="aa5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<4>" LOC="aa3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<3>" LOC="aa4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<2>" LOC="y3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<1>" LOC="y4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<0>" LOC="w3" | IOSTANDARD=LVDCI_33 | NODELAY;

NET "ram1_address<18>" LOC="ab3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<17>" LOC="ac5" | IOSTANDARD=LVDCI_33;
NET "ram1_address<16>" LOC="u6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<15>" LOC="v6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<14>" LOC="w6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<13>" LOC="y6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<12>" LOC="aa7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<11>" LOC="ab7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<10>" LOC="ac6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<9>" LOC="ad3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<8>" LOC="ad4" | IOSTANDARD=LVDCI_33;
NET "ram1_address<7>" LOC="u3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<6>" LOC="w4" | IOSTANDARD=LVDCI_33;
```

```
NET "ram1_address<5>" LOC="ac8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<4>" LOC="ab8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<3>" LOC="aa8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<2>" LOC="y7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<1>" LOC="y8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<0>" LOC="ad7" | IOSTANDARD=LVDCI_33;

NET "ram1_adv_ld" LOC="y5" | IOSTANDARD=LVDCI_33;
NET "ram1_clk" LOC="y9" | IOSTANDARD=LVDCI_33;
NET "ram1_cen_b" LOC="v5" | IOSTANDARD=LVDCI_33;
NET "ram1_ce_b" LOC="u4" | IOSTANDARD=LVDCI_33;
NET "ram1_oe_b" LOC="w5" | IOSTANDARD=LVDCI_33;
NET "ram1_we_b" LOC="aa6" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<0>" LOC="u2" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<1>" LOC="u1" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<2>" LOC="v1" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<3>" LOC="u5" | IOSTANDARD=LVDCI_33;

NET "clock_feedback_out" LOC="al28" | IOSTANDARD=LVDCI_33;
NET "clock_feedback_in" LOC="aj16";

#
# Flash
#

NET "flash_data<15>" LOC="ak10" | IOSTANDARD=LVTTL;
NET "flash_data<14>" LOC="ak11" | IOSTANDARD=LVTTL;
NET "flash_data<13>" LOC="ak12" | IOSTANDARD=LVTTL;
NET "flash_data<12>" LOC="ak13" | IOSTANDARD=LVTTL;
NET "flash_data<11>" LOC="ak14" | IOSTANDARD=LVTTL;
NET "flash_data<10>" LOC="ak15" | IOSTANDARD=LVTTL;
NET "flash_data<9>"  LOC="ah12" | IOSTANDARD=LVTTL;
NET "flash_data<8>"  LOC="ah13" | IOSTANDARD=LVTTL;
NET "flash_data<7>"  LOC="al10" | IOSTANDARD=LVTTL;
NET "flash_data<6>"  LOC="al11" | IOSTANDARD=LVTTL;
NET "flash_data<5>"  LOC="al12" | IOSTANDARD=LVTTL;
NET "flash_data<4>"  LOC="al13" | IOSTANDARD=LVTTL;
NET "flash_data<3>"  LOC="al14" | IOSTANDARD=LVTTL;
NET "flash_data<2>"  LOC="al15" | IOSTANDARD=LVTTL;
NET "flash_data<1>"  LOC="aj12" | IOSTANDARD=LVTTL;
NET "flash_data<0>"  LOC="aj13" | IOSTANDARD=LVTTL;

#NET "flash_address<24>" LOC="al7";
NET "flash_address<23>" LOC="aj15";
NET "flash_address<22>" LOC="al25";
NET "flash_address<21>" LOC="ak23";
NET "flash_address<20>" LOC="al23";
NET "flash_address<19>" LOC="ak22";
NET "flash_address<18>" LOC="al22";
NET "flash_address<17>" LOC="ak21";
NET "flash_address<16>" LOC="al21";
NET "flash_address<15>" LOC="ak20";
NET "flash_address<14>" LOC="al20";
NET "flash_address<13>" LOC="ak19";
NET "flash_address<12>" LOC="al19";
NET "flash_address<11>" LOC="al18";
NET "flash_address<10>" LOC="ak17";
NET "flash_address<9>"  LOC="al17";
NET "flash_address<8>"  LOC="ah21";
NET "flash_address<7>"  LOC="aj20";
NET "flash_address<6>"  LOC="ah20";
NET "flash_address<5>"  LOC="aj19";
NET "flash_address<4>"  LOC="ah19";
NET "flash_address<3>"  LOC="ah18";
NET "flash_address<2>"  LOC="aj17";
NET "flash_address<1>"  LOC="ae14";
NET "flash_address<0>"  LOC="ah14";

NET "flash_ce_b" LOC="aj21" | IOSTANDARD=LVDCI_33;

NET "flash_oe_b" LOC="ak9" | IOSTANDARD=LVDCI_33;
NET "flash_we_b" LOC="al8" | IOSTANDARD=LVDCI_33;
NET "flash_reset_b" LOC="ak18" | IOSTANDARD=LVDCI_33;
```

```
NET "flash_sts" LOC="al9" | PULLUP;
NET "flash_byte_b" LOC="ah15" | IOSTANDARD=LVDCI_33;

#
# RS-232
#

NET "rs232_txd" LOC="p4" | IOSTANDARD=LVDCI_33;
NET "rs232_rxd" LOC="p6";
NET "rs232_rts" LOC="r3" | IOSTANDARD=LVDCI_33;
NET "rs232_cts" LOC="n8";

#
# Mouse and Keyboard
#

NET "mouse_clock" LOC="ac16";
NET "mouse_data" LOC="ac15";
NET "keyboard_clock" LOC="ag16";
NET "keyboard_data" LOC="af16";

#
# Clocks
#

NET "clock_27mhz" LOC="c16";
NET "clock1" LOC="h16";
NET "clock2" LOC="c15";

#
# Alphanumeric Display
#

NET "disp_blank" LOC="af12" | IOSTANDARD=LVDCI_33;
NET "disp_data_in" LOC="ae12";
NET "disp_clock" LOC="af14" | IOSTANDARD=LVDCI_33;
NET "disp_rs" LOC="af15" | IOSTANDARD=LVDCI_33;
NET "disp_ce_b" LOC="af13" | IOSTANDARD=LVDCI_33;
NET "disp_reset_b" LOC="ag11" | IOSTANDARD=LVDCI_33;
NET "disp_data_out" LOC="ae15" | IOSTANDARD=LVDCI_33;

#
# Buttons and Switches
#

NET "button0" LOC="ae11";
NET "button1" LOC="ae10";
NET "button2" LOC="ad11";
NET "button3" LOC="ab12";
NET "button_enter" LOC="ak7";
NET "button_right" LOC="al6";
NET "button_left" LOC="al5";
NET "button_up" LOC="al4";
NET "button_down" LOC="ak6";

NET "switch<7>" LOC="ad22";
NET "switch<6>" LOC="ae23";
NET "switch<5>" LOC="ac20";
NET "switch<4>" LOC="ab20";
NET "switch<3>" LOC="ac21";
NET "switch<2>" LOC="ak25";
NET "switch<1>" LOC="al26";
NET "switch<0>" LOC="ak26";

#
# Discrete LEDs
#

NET "led<7>" LOC="ae17" | IOSTANDARD=LVTTL;
NET "led<6>" LOC="af17" | IOSTANDARD=LVTTL;
NET "led<5>" LOC="af18" | IOSTANDARD=LVTTL;
NET "led<4>" LOC="af19" | IOSTANDARD=LVTTL;
NET "led<3>" LOC="af20" | IOSTANDARD=LVTTL;
```

```
NET "led<2>" LOC="ag21" | IOSTANDARD=LVTTL;
NET "led<1>" LOC="ae21" | IOSTANDARD=LVTTL;
NET "led<0>" LOC="ae22" | IOSTANDARD=LVTTL;


#
# User Pins
#

NET "user1<31>" LOC="j15" | IOSTANDARD=LVTTL;
NET "user1<30>" LOC="j14" | IOSTANDARD=LVTTL;
NET "user1<29>" LOC="g15" | IOSTANDARD=LVTTL;
NET "user1<28>" LOC="f14" | IOSTANDARD=LVTTL;
NET "user1<27>" LOC="f12" | IOSTANDARD=LVTTL;
NET "user1<26>" LOC="h11" | IOSTANDARD=LVTTL;
NET "user1<25>" LOC="g9" | IOSTANDARD=LVTTL;
NET "user1<24>" LOC="h9" | IOSTANDARD=LVTTL;
NET "user1<23>" LOC="b15" | IOSTANDARD=LVTTL;
NET "user1<22>" LOC="b14" | IOSTANDARD=LVTTL;
NET "user1<21>" LOC="f15" | IOSTANDARD=LVTTL;
NET "user1<20>" LOC="e13" | IOSTANDARD=LVTTL;
NET "user1<19>" LOC="e11" | IOSTANDARD=LVTTL;
NET "user1<18>" LOC="e9" | IOSTANDARD=LVTTL;
NET "user1<17>" LOC="f8" | IOSTANDARD=LVTTL;
NET "user1<16>" LOC="f7" | IOSTANDARD=LVTTL;
NET "user1<15>" LOC="c13" | IOSTANDARD=LVTTL;
NET "user1<14>" LOC="c12" | IOSTANDARD=LVTTL;
NET "user1<13>" LOC="c11" | IOSTANDARD=LVTTL;
NET "user1<12>" LOC="c10" | IOSTANDARD=LVTTL;
NET "user1<11>" LOC="c9" | IOSTANDARD=LVTTL;
NET "user1<10>" LOC="c8" | IOSTANDARD=LVTTL;
NET "user1<9>" LOC="c6" | IOSTANDARD=LVTTL;
NET "user1<8>" LOC="e6" | IOSTANDARD=LVTTL;
NET "user1<7>" LOC="a11" | IOSTANDARD=LVTTL;
NET "user1<6>" LOC="a10" | IOSTANDARD=LVTTL;
NET "user1<5>" LOC="a9" | IOSTANDARD=LVTTL;
NET "user1<4>" LOC="a8" | IOSTANDARD=LVTTL;
NET "user1<3>" LOC="b6" | IOSTANDARD=LVTTL;
NET "user1<2>" LOC="b5" | IOSTANDARD=LVTTL;
NET "user1<1>" LOC="c5" | IOSTANDARD=LVTTL;
NET "user1<0>" LOC="b3" | IOSTANDARD=LVTTL;


NET "user2<31>" LOC="b27" | IOSTANDARD=LVTTL;
NET "user2<30>" LOC="b26" | IOSTANDARD=LVTTL;
NET "user2<29>" LOC="b25" | IOSTANDARD=LVTTL;
NET "user2<28>" LOC="a24" | IOSTANDARD=LVTTL;
NET "user2<27>" LOC="a23" | IOSTANDARD=LVTTL;
NET "user2<26>" LOC="a22" | IOSTANDARD=LVTTL;
NET "user2<25>" LOC="a21" | IOSTANDARD=LVTTL;
NET "user2<24>" LOC="a20" | IOSTANDARD=LVTTL;
NET "user2<23>" LOC="d26" | IOSTANDARD=LVTTL;
NET "user2<22>" LOC="d25" | IOSTANDARD=LVTTL;
NET "user2<21>" LOC="c24" | IOSTANDARD=LVTTL;
NET "user2<20>" LOC="d23" | IOSTANDARD=LVTTL;
NET "user2<19>" LOC="d21" | IOSTANDARD=LVTTL;
NET "user2<18>" LOC="d20" | IOSTANDARD=LVTTL;
NET "user2<17>" LOC="d19" | IOSTANDARD=LVTTL;
NET "user2<16>" LOC="d18" | IOSTANDARD=LVTTL;
NET "user2<15>" LOC="f24" | IOSTANDARD=LVTTL;
NET "user2<14>" LOC="f23" | IOSTANDARD=LVTTL;
NET "user2<13>" LOC="e22" | IOSTANDARD=LVTTL;
NET "user2<12>" LOC="e20" | IOSTANDARD=LVTTL;
NET "user2<11>" LOC="e18" | IOSTANDARD=LVTTL;
NET "user2<10>" LOC="e16" | IOSTANDARD=LVTTL;
NET "user2<9>" LOC="a19" | IOSTANDARD=LVTTL;
NET "user2<8>" LOC="a18" | IOSTANDARD=LVTTL;
NET "user2<7>" LOC="h22" | IOSTANDARD=LVTTL;
NET "user2<6>" LOC="g22" | IOSTANDARD=LVTTL;
NET "user2<5>" LOC="f21" | IOSTANDARD=LVTTL;
NET "user2<4>" LOC="f19" | IOSTANDARD=LVTTL;
NET "user2<3>" LOC="f17" | IOSTANDARD=LVTTL;
NET "user2<2>" LOC="h19" | IOSTANDARD=LVTTL;
```

```
NET "user2<1>" LOC="g20" | IOSTANDARD=LVTTL;
NET "user2<0>" LOC="h21" | IOSTANDARD=LVTTL;

NET "user3<31>" LOC="g12" | IOSTANDARD=LVTTL;
NET "user3<30>" LOC="h13" | IOSTANDARD=LVTTL;
NET "user3<29>" LOC="j13" | IOSTANDARD=LVTTL;
NET "user3<28>" LOC="g14" | IOSTANDARD=LVTTL;
NET "user3<27>" LOC="f13" | IOSTANDARD=LVTTL;
NET "user3<26>" LOC="f11" | IOSTANDARD=LVTTL;
NET "user3<25>" LOC="g10" | IOSTANDARD=LVTTL;
NET "user3<24>" LOC="h10" | IOSTANDARD=LVTTL;
NET "user3<23>" LOC="a15" | IOSTANDARD=LVTTL;
NET "user3<22>" LOC="a14" | IOSTANDARD=LVTTL;
NET "user3<21>" LOC="e15" | IOSTANDARD=LVTTL;
NET "user3<20>" LOC="e14" | IOSTANDARD=LVTTL;
NET "user3<19>" LOC="e12" | IOSTANDARD=LVTTL;
NET "user3<18>" LOC="e10" | IOSTANDARD=LVTTL;
NET "user3<17>" LOC="f9" | IOSTANDARD=LVTTL;
NET "user3<16>" LOC="g8" | IOSTANDARD=LVTTL;
NET "user3<15>" LOC="d14" | IOSTANDARD=LVTTL;
NET "user3<14>" LOC="d13" | IOSTANDARD=LVTTL;
NET "user3<13>" LOC="d12" | IOSTANDARD=LVTTL;
NET "user3<12>" LOC="d11" | IOSTANDARD=LVTTL;
NET "user3<11>" LOC="d9" | IOSTANDARD=LVTTL;
NET "user3<10>" LOC="d8" | IOSTANDARD=LVTTL;
NET "user3<9>" LOC="d7" | IOSTANDARD=LVTTL;
NET "user3<8>" LOC="d6" | IOSTANDARD=LVTTL;
NET "user3<7>" LOC="b12" | IOSTANDARD=LVTTL;
NET "user3<6>" LOC="b11" | IOSTANDARD=LVTTL;
NET "user3<5>" LOC="b10" | IOSTANDARD=LVTTL;
NET "user3<4>" LOC="b9" | IOSTANDARD=LVTTL;
NET "user3<3>" LOC="a7" | IOSTANDARD=LVTTL;
NET "user3<2>" LOC="a6" | IOSTANDARD=LVTTL;
NET "user3<1>" LOC="a5" | IOSTANDARD=LVTTL;
NET "user3<0>" LOC="a4" | IOSTANDARD=LVTTL;

NET "user4<31>" LOC="a28" | IOSTANDARD=LVTTL;
NET "user4<30>" LOC="a27" | IOSTANDARD=LVTTL;
NET "user4<29>" LOC="a26" | IOSTANDARD=LVTTL;
NET "user4<28>" LOC="a25" | IOSTANDARD=LVTTL;
NET "user4<27>" LOC="b23" | IOSTANDARD=LVTTL;
NET "user4<26>" LOC="b22" | IOSTANDARD=LVTTL;
NET "user4<25>" LOC="b21" | IOSTANDARD=LVTTL;
NET "user4<24>" LOC="b20" | IOSTANDARD=LVTTL;
NET "user4<23>" LOC="e25" | IOSTANDARD=LVTTL;
NET "user4<22>" LOC="c26" | IOSTANDARD=LVTTL;
NET "user4<21>" LOC="d24" | IOSTANDARD=LVTTL;
NET "user4<20>" LOC="c23" | IOSTANDARD=LVTTL;
NET "user4<19>" LOC="c22" | IOSTANDARD=LVTTL;
NET "user4<18>" LOC="c21" | IOSTANDARD=LVTTL;
NET "user4<17>" LOC="c20" | IOSTANDARD=LVTTL;
NET "user4<16>" LOC="c19" | IOSTANDARD=LVTTL;
NET "user4<15>" LOC="g24" | IOSTANDARD=LVTTL;
NET "user4<14>" LOC="e24" | IOSTANDARD=LVTTL;
NET "user4<13>" LOC="e23" | IOSTANDARD=LVTTL;
NET "user4<12>" LOC="e21" | IOSTANDARD=LVTTL;
NET "user4<11>" LOC="e19" | IOSTANDARD=LVTTL;
NET "user4<10>" LOC="e17" | IOSTANDARD=LVTTL;
NET "user4<9>" LOC="b19" | IOSTANDARD=LVTTL;
NET "user4<8>" LOC="b18" | IOSTANDARD=LVTTL;
NET "user4<7>" LOC="h23" | IOSTANDARD=LVTTL;
NET "user4<6>" LOC="g23" | IOSTANDARD=LVTTL;
NET "user4<5>" LOC="g21" | IOSTANDARD=LVTTL;
NET "user4<4>" LOC="f20" | IOSTANDARD=LVTTL;
NET "user4<3>" LOC="f18" | IOSTANDARD=LVTTL;
NET "user4<2>" LOC="f16" | IOSTANDARD=LVTTL;
NET "user4<1>" LOC="g18" | IOSTANDARD=LVTTL;
NET "user4<0>" LOC="g17" | IOSTANDARD=LVTTL;

#
# Daughter Card
#
```

```
NET "daughtercard<43>" LOC="L7"  | IOSTANDARD=LVTTL;
NET "daughtercard<42>" LOC="H1"  | IOSTANDARD=LVTTL;
NET "daughtercard<41>" LOC="J2"  | IOSTANDARD=LVTTL;
NET "daughtercard<40>" LOC="J1"  | IOSTANDARD=LVTTL;
NET "daughtercard<39>" LOC="K2"  | IOSTANDARD=LVTTL;
NET "daughtercard<38>" LOC="M7"  | IOSTANDARD=LVTTL;
NET "daughtercard<37>" LOC="M6"  | IOSTANDARD=LVTTL;
NET "daughtercard<36>" LOC="M3"  | IOSTANDARD=LVTTL;
NET "daughtercard<35>" LOC="M4"  | IOSTANDARD=LVTTL;
NET "daughtercard<34>" LOC="L3"  | IOSTANDARD=LVTTL;
NET "daughtercard<33>" LOC="K1"  | IOSTANDARD=LVTTL;
NET "daughtercard<32>" LOC="L4"  | IOSTANDARD=LVTTL;
NET "daughtercard<31>" LOC="K3"  | IOSTANDARD=LVTTL;
NET "daughtercard<30>" LOC="K9"  | IOSTANDARD=LVTTL;
NET "daughtercard<29>" LOC="L9"  | IOSTANDARD=LVTTL;
NET "daughtercard<28>" LOC="K8"  | IOSTANDARD=LVTTL;
NET "daughtercard<27>" LOC="K7"  | IOSTANDARD=LVTTL;
NET "daughtercard<26>" LOC="L8"  | IOSTANDARD=LVTTL;
NET "daughtercard<25>" LOC="L6"  | IOSTANDARD=LVTTL;
NET "daughtercard<24>" LOC="M5"  | IOSTANDARD=LVTTL;
NET "daughtercard<23>" LOC="N5"  | IOSTANDARD=LVTTL;
NET "daughtercard<22>" LOC="P5"  | IOSTANDARD=LVTTL;
NET "daughtercard<21>" LOC="D3"  | IOSTANDARD=LVTTL;
NET "daughtercard<20>" LOC="E4"  | IOSTANDARD=LVTTL;
NET "daughtercard<19>" LOC="E3"  | IOSTANDARD=LVTTL;
NET "daughtercard<18>" LOC="F4"  | IOSTANDARD=LVTTL;
NET "daughtercard<17>" LOC="F3"  | IOSTANDARD=LVTTL;
NET "daughtercard<16>" LOC="G4"  | IOSTANDARD=LVTTL;
NET "daughtercard<15>" LOC="H4"  | IOSTANDARD=LVTTL;
NET "daughtercard<14>" LOC="J3"  | IOSTANDARD=LVTTL;
NET "daughtercard<13>" LOC="J4"  | IOSTANDARD=LVTTL;
NET "daughtercard<12>" LOC="D2"  | IOSTANDARD=LVTTL;
NET "daughtercard<11>" LOC="D1"  | IOSTANDARD=LVTTL;
NET "daughtercard<10>" LOC="E2"  | IOSTANDARD=LVTTL;
NET "daughtercard<9>" LOC="E1"  | IOSTANDARD=LVTTL;
NET "daughtercard<8>" LOC="F5"  | IOSTANDARD=LVTTL;
NET "daughtercard<7>" LOC="G5"  | IOSTANDARD=LVTTL;
NET "daughtercard<6>" LOC="H5"  | IOSTANDARD=LVTTL;
NET "daughtercard<5>" LOC="J5"  | IOSTANDARD=LVTTL;
NET "daughtercard<4>" LOC="K5"  | IOSTANDARD=LVTTL;
NET "daughtercard<3>" LOC="H7"  | IOSTANDARD=LVTTL;
NET "daughtercard<2>" LOC="J8"  | IOSTANDARD=LVTTL;
NET "daughtercard<1>" LOC="J6"  | IOSTANDARD=LVTTL;
NET "daughtercard<0>" LOC="J7"  | IOSTANDARD=LVTTL;

#
# System Ace
#

NET "systemace_data<15>" LOC="F29"  | IOSTANDARD=LVTTL;
NET "systemace_data<14>" LOC="G28"  | IOSTANDARD=LVTTL;
NET "systemace_data<13>" LOC="H29"  | IOSTANDARD=LVTTL;
NET "systemace_data<12>" LOC="H28"  | IOSTANDARD=LVTTL;
NET "systemace_data<11>" LOC="J29"  | IOSTANDARD=LVTTL;
NET "systemace_data<10>" LOC="J28"  | IOSTANDARD=LVTTL;
NET "systemace_data<9>" LOC="K29"  | IOSTANDARD=LVTTL;
NET "systemace_data<8>" LOC="L29"  | IOSTANDARD=LVTTL;
NET "systemace_data<7>" LOC="L28"  | IOSTANDARD=LVTTL;
NET "systemace_data<6>" LOC="M29"  | IOSTANDARD=LVTTL;
NET "systemace_data<5>" LOC="M28"  | IOSTANDARD=LVTTL;
NET "systemace_data<4>" LOC="N29"  | IOSTANDARD=LVTTL;
NET "systemace_data<3>" LOC="N28"  | IOSTANDARD=LVTTL;
NET "systemace_data<2>" LOC="P28"  | IOSTANDARD=LVTTL;
NET "systemace_data<1>" LOC="R29"  | IOSTANDARD=LVTTL;
NET "systemace_data<0>" LOC="R28"  | IOSTANDARD=LVTTL;

NET "systemace_address<6>" LOC="E29"  | IOSTANDARD=LVTTL;
NET "systemace_address<5>" LOC="F28"  | IOSTANDARD=LVTTL;
NET "systemace_address<4>" LOC="H31"  | IOSTANDARD=LVTTL;
NET "systemace_address<3>" LOC="J30"  | IOSTANDARD=LVTTL;
NET "systemace_address<2>" LOC="J31"  | IOSTANDARD=LVTTL;
NET "systemace_address<1>" LOC="K30"  | IOSTANDARD=LVTTL;
NET "systemace_address<0>" LOC="K31"  | IOSTANDARD=LVTTL;
```

```
NET "systemace_ce_b" LOC="E28" | IOSTANDARD=LVTTL;
NET "systemace_we_b" LOC="P31" | IOSTANDARD=LVTTL;
NET "systemace_oe_b" LOC="R31" | IOSTANDARD=LVTTL;
NET "systemace_irq" LOC="D29";
NET "systemace_mpbrdy" LOC="L31";

#
# Logic Analyzer
#

NET "analyzer1_data<15>" LOC="G1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<14>" LOC="H3" | IOSTANDARD=LVTTL;
NET "analyzer1_data<13>" LOC="M9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<12>" LOC="M8" | IOSTANDARD=LVTTL;
NET "analyzer1_data<11>" LOC="L5" | IOSTANDARD=LVTTL;
NET "analyzer1_data<10>" LOC="L1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<9>" LOC="L2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<8>" LOC="N9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<7>" LOC="M1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<6>" LOC="M2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<5>" LOC="N1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<4>" LOC="N2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<3>" LOC="P1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<2>" LOC="P2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<1>" LOC="R1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<0>" LOC="R2" | IOSTANDARD=LVTTL;
NET "analyzer1_clock" LOC="G2" | IOSTANDARD=LVTTL;

NET "analyzer2_data<15>" LOC="f2" | IOSTANDARD=LVTTL;
NET "analyzer2_data<14>" LOC="k10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<13>" LOC="l10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<12>" LOC="m10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<11>" LOC="r7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<10>" LOC="n3" | IOSTANDARD=LVTTL;
NET "analyzer2_data<9>" LOC="r8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<8>" LOC="r9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<7>" LOC="p9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<6>" LOC="n6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<5>" LOC="p7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<4>" LOC="n4" | IOSTANDARD=LVTTL;
NET "analyzer2_data<3>" LOC="t8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<2>" LOC="t9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<1>" LOC="r6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<0>" LOC="r5" | IOSTANDARD=LVTTL;
NET "analyzer2_clock" LOC="f1" | IOSTANDARD=LVTTL;

NET "analyzer3_data<15>" LOC="k24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<14>" LOC="k25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<13>" LOC="k22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<12>" LOC="l24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<11>" LOC="l25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<10>" LOC="l22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<9>" LOC="l23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<8>" LOC="m23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<7>" LOC="m24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<6>" LOC="m25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<5>" LOC="k23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<4>" LOC="m22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<3>" LOC="n23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<2>" LOC="p23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<1>" LOC="r23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<0>" LOC="r24" | IOSTANDARD=LVTTL;
NET "analyzer3_clock" LOC="j24" | IOSTANDARD=LVTTL;

NET "analyzer4_data<15>" LOC="ag7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<14>" LOC="ak3" | IOSTANDARD=LVTTL;
NET "analyzer4_data<13>" LOC="aj5" | IOSTANDARD=LVTTL;
NET "analyzer4_data<12>" LOC="ak29" | IOSTANDARD=LVTTL;
NET "analyzer4_data<11>" LOC="ak28" | IOSTANDARD=LVTTL;
NET "analyzer4_data<10>" LOC="af25" | IOSTANDARD=LVTTL;
NET "analyzer4_data<9>" LOC="ag24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<8>" LOC="af24" | IOSTANDARD=LVTTL;
```

```
NET "analyzer4_data<7>" LOC="af23" | IOSTANDARD=LVTTL;
NET "analyzer4_data<6>" LOC="al27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<5>" LOC="ak27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<4>" LOC="ah17" | IOSTANDARD=LVTTL;
NET "analyzer4_data<3>" LOC="ad13" | IOSTANDARD=LVTTL;
NET "analyzer4_data<2>" LOC="v7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<1>" LOC="u7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<0>" LOC="u8" | IOSTANDARD=LVTTL;
NET "analyzer4_clock" LOC="ad9" | IOSTANDARD=LVTTL;
```